Winter 2018

Real-time Planning as Decision-making Under Uncertainty

Andrew James Mitchell
University of New Hampshire, Durham

REAL-TIME PLANNING AS DECISION-MAKING UNDER UNCERTAINTY

BY

ANDREW MITCHELL

BS Computer Science, University of New Hampshire, 2017

THESIS

Submitted to the University of New Hampshire
in Partial Fulfillment of
the Requirements for the Degree of

Master of Science

in

Computer Science

December, 2018

This Thesis was examined and approved in partial fulfillment of the requirements for the degree of Master of Science in Computer Science by:

Thesis director, Wheeler Ruml
Professor of Computer Science

Marek Petrik
Assistant Professor of Computer Science

Jörg Hoffmann
Professor of Computer Science, Saarland University

On December 3, 2018

Approval signatures are on file with the University of New Hampshire Graduate School.

TABLE OF CONTENTS

LIST OF TABLES iv

LIST OF FIGURES vi

ABSTRACT vii

Chapter 0 Introduction **1**

0.1 Previous Work 3

Chapter 1 Decision-making from Lookahead **6**

1.1 Theory 6

1.2 Experiments 10

1.3 Implementation 15

Chapter 2 Choosing Nodes to Expand **34**

2.1 Theory 34

2.2 Experiments 36

2.3 Implementation 38

Chapter 3 Conclusion **45**

3.1 Limitations 45

3.2 Contributions 46

Bibliography **47**

LIST OF TABLES

1-1 Random Tree ϵ Values With Depth-First Expansion 27

1-2 Random Tree ϵ Values With A* Expansion 30

LIST OF FIGURES

0-1 Should an agent at A move to B_1 or B_2? 2

0-2 Should an agent expand nodes under α or β? 2

1-1 Backup rules with one level unknown. 11

1-2 Backup rules with one level unknown. Violin plots show distribution of the data as well as a box plot with the median and quartiles. The red horizontal lines depict mean while red vertical lines show the 95% confidence intervals. 12

1-3 Backup rules with depth-first lookahead on trees. 13

1-4 Backup rules with depth-first lookahead on trees. Violin plots show distribution of the data as well as a box plot with the median and quartiles. The red horizontal lines depict mean while red vertical lines show the 95% confidence intervals. . . . 14

1-5 Backup rules with A* lookahead on trees. 15

1-6 Backup rules with A* lookahead on trees. Violin plots show distribution of the data as well as a box plot with the median and quartiles. The red horizontal lines depict mean while red vertical lines show the 95% confidence intervals. . . . 16

1-7 Backup rules with A* lookahead on 15 puzzles. 17

1-8 Backup rules with A* lookahead on 15 puzzles. Violin plots show distribution of the data as well as a box plot with the median and quartiles. The red horizontal lines depict mean while red vertical lines show the 95% confidence intervals. . . . 18

1-9 Obtaining the beliefs at the leaves as described in (Pemberton, 1995). 20

1-10 CDF of the belief at the leaves described in (Pemberton, 1995). 21

1-11 CDF of the belief that is obtained from the Cserna backup of two nodes using beliefs from (Pemberton, 1995). 24

1-12 Belief obtained from performing Cserna backup on two nodes using generalized beliefs based around $\hat{f}$. 25

1-13 Belief that has been squished to estimate how additional search will effect the belief at the TLA. 26
1-14 Pilot experiment results on random binary tree of depth 100 with A* expansion. 28
1-15 Pilot experiment results on random binary tree of depth 100 with depth-first expansion. 29

2-1 Expansion strategies on random trees. 37
2-2 Expansion strategies on random trees. Violin plots show distribution of the data as well as a box plot with the median and quartiles. The red horizontal lines depict mean while red vertical lines show the 95% confidence intervals. 38
2-3 Expansion strategies on 15 puzzles. 39
2-4 Expansion strategies on 15 puzzles. Violin plots show distribution of the data as well as a box plot with the median and quartiles. The red horizontal lines depict mean while red vertical lines show the 95% confidence intervals. 40
2-5 Optimality gap of expansion strategies on 15 puzzles as a percentage of A*'s. . . 41

ABSTRACT

REAL-TIME PLANNING AS DECISION-MAKING UNDER UNCERTAINTY

by

ANDREW MITCHELL

University of New Hampshire, December, 2018

In real-time planning, an agent must select the next action to take within a fixed time bound. Many popular real-time heuristic search methods approach this by expanding nodes using time-limited A* and selecting the action leading toward the frontier node with the lowest f value. In this thesis, we reconsider real-time planning as a problem of decision-making under uncertainty. We treat heuristic values as uncertain evidence and we explore several backup methods for aggregating this evidence. We then propose a novel lookahead strategy that expands nodes to minimize risk, the expected regret in case a non-optimal action is chosen. We evaluate these methods in a simple synthetic benchmark and the sliding tile puzzle and find that they outperform previous methods. This work illustrates how uncertainty can arise even when solving deterministic planning problems, due to the inherent ignorance of time-limited search algorithms about those portions of the state space that they have not computed, and how an agent can benefit from explicitly meta-reasoning about this uncertainty.

CHAPTER 0

Introduction

In some AI applications, such as user interfaces or fixed-wing aircraft control, it is undesirable for the system to exhibit unbounded pauses between actions. In real-time planning, an agent is required to select its next action within a fixed time bound. The agent attempts to minimize its total trajectory cost as it incrementally plans toward a goal. Many real-time heuristic search methods have been developed for this problem setting. Many of them follow the basic three-phase paradigm set down in the seminal work of Korf (1990): 1) starting at the agent's current state, expand a fixed number of nodes to form a lookahead search space (LSS), 2) use the heuristic values of the frontier nodes in combination with the path costs incurred to reach them to estimate the cost-to-goal the agent would incur if it selected each possible currently-applicable action, and then 3) commit to the lowest cost action and, to prevent the agent from cycling if it returns to the same state in the future, update the heuristic values of one or more states in the LSS. For example, in the popular and typical algorithm LSS-LRTA* (Koenig & Sun, 2008), the lookahead in step one is performed using A* (Hart, Nilsson, & Raphael, 1968), the value estimates in step two are implicitly calculated for each node as the minimum f value among its successors (the 'minimin' backup), and the learning in step three is performed by updating h values for all nodes in the LSS using a variant of Dijkstra's algorithm.

However, this paradigm is not necessarily optimal. This becomes apparent when one considers the problem from the perspective of decision-making under uncertainty. For example, Pemberton and Korf (1994) observed that, given an LSS, the optimal decision for the agent is not, in general, to head toward the frontier node with the lowest f value. Figure 0-1 shows their example LSS, in which an agent located at node A must decide whether to transition to node B_1 or B_2. All nodes at depth three after the C_i are goals. Recall that the principle of rationality demands that an

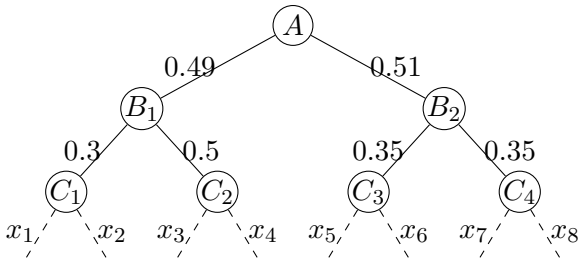

Figure 0-1: Should an agent at A move to B_1 or B_2?

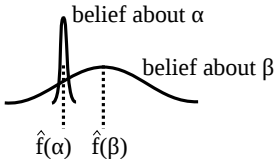

Figure 0-2: Should an agent expand nodes under α or β?

agent take an action that minimizes expected cost. All subsequent edge costs x_i are unknown but uniformly distributed between 0 and 1, so $h = 0$ for all nodes C_i. C_1 is the node of lowest f (namely 0.79) so a typical real-time search would move to B_1. However, moving to B_2 minimizes the agent's expected cost, because in the next iteration the costs $x_5, ..., x_8$ will be revealed and the expected minimum of these four values (0.2) yields a total expected cost for B_2 (1.06) less than that of B_1 (1.066, see Pemberton, (1995), eq. 1 for details). In other words, having many good-looking options can be statistically better than a single great-looking option. This problem does not arise in the off-line planning setting, as there the agent can discover all relevant edge costs before committing to an action.

Furthermore, it was noted by Mutchler (1986) that A*'s policy of expanding the frontier node with the lowest f value is not, in general, the optimal way to make use of a limited number of node expansions. If we view the agent as facing a decision about which currently-applicable action is best, it is sometimes beneficial to gain knowledge by expanding a different node.

For a simple example, consider the situation depicted in Figure 0-2, in which we represent the agent's current belief about the remaining plan cost that will be incurred by committing to an action by a probability distribution over possible costs. The expected value is denoted by $\hat{f}(\cdot)$.

If the agent is quite certain about the value of action α but quite uncertain about the value of a different action β, then expanding frontier nodes under α can be less useful than expanding under β, even if β is believed to have a higher expected cost. In other words, it can be more important to explore the possibility that a poorly-understood option might in fact be great than to nail down the exact value of a well-understood good-looking option. This problem does not arise in off-line optimal search, in which every node whose f value is less than the optimal solution cost must be expanded.

Although these insights seem powerful, they have not yet resulted in generally-applicable real-time search algorithms. In this thesis, we advance toward practical algorithms that realize the reasoning under uncertainty perspective. We study strategies for expanding nodes and methods for backing up their information for decision making, culminating in a novel general-purpose method for real-time search that we call Nancy. Experimental results in random trees and in the sliding tile puzzle suggest that Nancy makes better use of limited node expansions than conventional real-time search algorithms. While additional work will be required to engineer an optimized implementation, Nancy already shows how reasoning about uncertainty can play an important role in resource-constrained search even for deterministic domains.

0.1 Previous Work

Mutchler (1986) raises the question of how best to allocate a limited number of expansions. His analysis considers complete binary trees of uniform depth where each edge is randomly assigned cost 1 with probability p and cost 0 otherwise. He proved that a minimum f expansion policy is not optimal for such trees in general, but that it is optimal for certain values of p and certain numbers of expansions. It is not clear how to apply these results to more realistic state spaces.

Pemberton and Korf (1994) point out that it can be useful to use different criteria for action selection versus node expansion. They use binary trees with random edge costs uniformly distributed between 0 and 1 and use a computer algebra package to generate code to compute exact $\hat{f}$ values under the assumption that only one or two tree levels remain until a goal (they call this the 'last incremental decision problem'). As we will discuss in more detail below, this requires representing

and reasoning about the distribution of possible values under child nodes in order to compute the distribution at each parent node. They conclude that a strategy based on expected values is barely better than the classic minimin strategy and impractical to compute for state spaces beyond tiny trees. They also investigate a method in which the nodes with minimum f are expanded and the action with minimum $\hat{f}$ is selected and find that it performs better than using f for both.

Given the pessimism surrounding exact estimates, Pemberton (1995) proposes an approximate method called k-best. Only the k best frontier nodes below each top-level action are used to compute their expected values, allowing a fixed inventory of equations derived in advance to be used to compute expected values during search. Although this approach did surpass minimin in experiments on random binary trees, Pemberton concludes that its complexity makes it impractical. It is also not clear how to apply these results beyond random binary trees.

Our problem setting bears a superficial similarity to the exploration/exploitation trade-off examined in reinforcement learning. However, note that our central challenge is how to make use of a given number of expansions — we do not have to decide between exploring for more information (by expanding additional nodes) or exploiting our current estimates (by committing to the currently-best-looking action). DTA* (Russell & Wefald, 1991) and Mo'RTS (O'Ceallaigh & Ruml, 2015) are examples of real-time search algorithms that directly address that trade-off. Both are based on estimating the value of the information potentially gained by additional lookahead search and comparing this to a time penalty for the delay incurred. DTA* expands the frontier node with minimum f and Mo'RTS expands the frontier node with minimum $\hat{f}$. Furthermore, Mo'RTS models the potential path costs through a frontier node as a belief distribution centered around $\hat{f}$, and assumes that any additional search will shrink the variance of this belief. Mo'RTS uses these beliefs to determine if it is beneficial to take an identity action to allow for more search time.

Slo'RTS (Cserna, Ruml, & Frank, 2017) is another algorithm, like Mo'RTS, that decides if it is worth trading goal achievement time for additional search. However, Slo'RTS allows this to be done in domains where agents cannot take identity actions. Instead, the agent is allowed to execute sub-optimal actions to give it additional time in the next search iteration. The agent essentially slows itself down so that it can perform additional search.

MCTS algorithms such as UCT (Kocsis & Szepesvári, 2006) share our motivation of recognizing the uncertainty in the agent's beliefs and trying to generate relevant parts of the state space. However, our domain is deterministic and we have no need to sample action transitions or perform roll-outs. Furthermore, real-time planning can arise in applications where perhaps only a dozen nodes can be generated per decision, a regime where MCTS algorithms can perform poorly.

Tolpin and Shimony (2012) guide MCTS rollouts such that regret is minimized, a metric which is similar to our risk metric. They use a 'value of information'-inspired approach, using different sampling methods at the root of their rollouts. They find that using an ϵ-greedy or $\text{UCB}_{\sqrt{\cdot}}$ policy to sample the root and UCT for the remainder of the rollout implicitly minimizes regret better than UCB or UCT methods would alone. However, this is done only at the root of their rollouts and their method does not explicitly estimate regret in the same way that we estimate risk. It should also be noted that a single one of their rollouts could expand dozens or hundreds of nodes, which as pointed out above, could lead to poor performance in situations where expansions are limited.

Work on active learning also emphasizes careful selection of computations to refine beliefs. For example, Frazier, Powell, and Dayanik (2008) present an approach they term 'the knowledge gradient' for allocating measurements subject to noise in order to maximize decision quality. More broadly, the notion of representing beliefs over values during learning and decision-making has been pursued in Bayesian reinforcement learning (Bellemare, Dabney, & Munos, 2017, and references therein).

In the same vein as this thesis, Spaniol (2018) explored alternative expansion methods in the context of domain independent planning. Specifically, his thesis focused on implementing confidence based expansion with the fast downward planner. Confidence can be seen as an alternative but related metric to risk, however, it ignores the expected costs of beliefs at the frontier. This is discussed further in chapter 2, section 1.

This thesis is largely based on work that is published for AAAI-19: Andrew Mitchell, Wheeler Ruml, Fabian Spaniol, Jörg Hoffmann, and Marek Petrik, "Real-time Planning as Decision-making Under Uncertainty" (2019). I was responsible for the first draft of this paper, as well as all of the experimental results. The source code, written in C++, is available on GitHub (Mitchell, 2018).

CHAPTER 1

Decision-making from Lookahead

Real-time planning requires us to commit to one of the actions applicable at the agent's current state — we call these top-level actions (TLAs). The assumption behind lookahead is that the f values of deeper nodes are more informed than those of the current state's immediate successors. Our belief about the total trajectory cost we will be able to achieve by moving through a node n depends on our beliefs about n's successors. Therefore, after generating an LSS, we want to somehow back up the frontier f values up the tree through their ancestors to estimate the value of each TLA. In this section, we identify and explore four distinct backup rules, each with its own assumptions about the unexplored portion of the state space and the behavior of the agent: minimin, Bellman, Nancy, and Cserna. Using these, we will also see how to generalize Pemberton (1995)'s k-best method to arbitrary trees.

1.1 Theory

The following section will cover the theory behind each of the identified backup rules. Experimental results involving these rules will then be discussed, followed by some implementation details.

1.1.1 Minimin Backups

Minimin has been used in real-time search since the seminal work of Korf (1990). It assigns a parent node p the minimum f among its successors $S(p)$:

$$f(p) = \min_{c \in S(p)} f(c)$$

The best path to a goal must go through a successor, so the parent's f value must be at least as large as its best child's. It is used implicitly in the learning phase of LSS-LRTA*, as adjusting

states' h values so as to satisfy the minimin equation raises them as high as possible while preserving admissibility (Koenig & Sun, 2008). While this is a desirable property for h learning, we noted earlier that f is a lower bound rather than an expected cost, and is thus not appropriate as a basis for rational action selection. Minimin becomes rational and optimal when there is no uncertainty and the frontier f values are equal to the true f^* values. While this case is sufficiently approximated by the end of an A* search, it does not accurately model real-time planning.

1.1.2 Bellman Backups

A Bellman backup (Bellman, 1957) explicitly estimates expected value, again as the minimum over successors:

$$\hat{f}(p) = \min_{c \in S(p)} \hat{f}(c)$$

By using expected value, the Bellman backup recognizes the uncertainty that remains between states at the search frontier and any goals lying beyond. It is used as the basis of action selection by Pemberton and Korf (1994) and for both action selection and node expansion by Kiesel, Burns, and Ruml (2015). (In its usual form with stochastic transitions, it is of course also the basis for value iteration methods for MDPs.)

The Bellman backup has two weaknesses. First, it conveys only a scalar expected value. While this is all that is needed for selecting a TLA, we saw in Figure 0-2 and will see again below that having a complete belief distribution available is useful both for more sophisticated backups and for guiding lookahead expansions. The second weakness is that it assumes that no additional information will become available as the agent traverses the LSS. In real-time search, by the time the agent reaches a node deeper in the tree, further lookahead will have produced additional information that can inform its choices. In Figure 0-1, for example, Bellman will backup the expected value of C_3 (or C_4) to B_2, ignoring the fact that by the time the agent makes a decision at B_2, it will have the best of $x_5, ..., x_8$ to choose from, rather than just the values below C_3 (or C_4). Backing up from merely one child does not capture what the agent can expect to achieve. This raises a subtle point that, to our knowledge, has not been addressed in previous work: there is a distinction between the lowest cost of any solution through a node $f^*(n)$, which is independent of the agent's resources and

planning algorithm, and the cost that a resource-bounded agent can actually expect to achieve by moving through n, which we notate $f^{@}$. The latter depends on many factors, including lookahead budget, expansion strategy, and backup rule, and will likely be higher than f^* unless we have full knowledge of the relevant remaining state space. While off-line planners compute f^*, we need to take the agent's information state into account and aim to estimate $f^{@}$.

1.1.3 Nancy Backups

To address the first weakness of Bellman backups, lack of a full belief distribution, we introduce the Nancy backup, which assigns to the parent the belief of the child with the lowest expected cost:

$$B(p) \in \underset{d \in \{B(c) | c \in S(p)\}}{\operatorname{argmin}} \mathbb{E}(d)$$

Of course, this requires that we define the agent's belief about $f^{@}$ at every frontier node. This is not hard (O'Ceallaigh & Ruml, 2015) and we will see examples below. While Nancy backups are very similar to Bellman backups, we will see below that having full belief distributions can yield advantages. They do share Bellman's weakness of assuming that no more information will become available. As such, they are only correct if this assumption holds or if the best frontier node under a TLA is so much better than all of the others that its belief is disjoint from theirs.

1.1.4 Cserna Backups

The second weakness of Bellman backups for real-time search, insensitivity to additional information, was pointed out by Cserna et al. (2017). Although they did not formulate it explicitly, their work implies a new backup rule that assumes that, by the time the agent reaches a node, it will know the true value of each successor and be able to choose the minimum among them. Thus the probability we assign to a value for the parent corresponds to the probability that it will be the minimum of all the successor's values.

$$\mathbb{P}\left[f^{@}(p) \leq x\right] = \mathbb{P}\left[(\min_{c \in S(p)} f^{@}(c)) \leq x\right] \tag{1.1}$$

We assume that the probability of a configuration of values for the children follows our current beliefs about them and, in our implementation, that the beliefs of the successors are independent.

We note that the Cserna backup is associative — performing a Cserna backup directly on all the frontier nodes beneath a TLA will result in the same belief as performing Cserna backups recursively up the tree from the frontier to the TLA:

$$\begin{aligned}\mathbb{P}\left[f^{@}(p) \leq x\right] &= \mathbb{P}\left[\min_{c \in S(p)} \min_{d \in S(c)} f^{@}(d) \leq x\right] \\ &= \mathbb{P}\left[\min_{d \in S(S(p))} f^{@}(d) \leq x\right] \\ &= \mathbb{P}\left[\min_{d \in S(...S(p))} f^{@}(d) \leq x\right]\end{aligned}$$

where the set-valued S is defined as $S(A) = \bigcup_{a \in A} S(a)$.

Because they take into account the distributions of all children under a node, Cserna backups will make the correct decision in the Figure 0-1, moving to B_2 because of the high chance of a low value among the $x_5, ..., x_8$ rather than to B_1 toward the frontier node with minimum f. In other words, the expected value of a Cserna backup can be lower than the expected value of the best child distribution, which can lead to different action selection than Bellman backups.

Cserna backups assume that all of the information about the frontier will be available when it is reached by the agent. However, in the case of real-time search, this assumption only holds if the lookahead is large enough to exhaust the remaining reachable state space on the next iteration. Otherwise, Cserna backups will be optimistic about a node's value. Spaniol (2018) also finds that using Cserna backups results in preference for nodes with higher branching factor over nodes with lower branching factor.

1.1.5 The k-Best Strategy

With these backup strategies defined, it becomes straightforward to generalize Pemberton (1995)'s k-best method to arbitrary trees. His closed-form analysis of the last incremental decision problem can be seen as a specialization of Cserna backups to his random trees. We will define the k 'best' frontier nodes as those having the lowest $\hat{f}$ values. We backup all frontier nodes using Nancy backups, except when two or more of the best meet at the same parent, in which case we use a Cserna backup on them. We now consider this parent one of the 'best' and continue up the tree. This method interpolates between Nancy backups ($k = 1$) and Cserna backups ($k \geq$ maximum

number of frontier nodes under any TLA) while limiting the number of expensive Cserna backups to at most $k-1$ under each TLA.

1.2 Experiments

While each of the backup strategies has cases in which it is optimal, none of these correspond to actual real-time search. In this section, we experimentally evaluate the backup rules to see which perform best in practice. We use as benchmarks both random trees (following Pemberton and Korf (1994)) and the classic 15 puzzle benchmark (following Korf (1990)). The random trees were generated lazily yet deterministically, with branching factor 2 and edge costs uniformly distributed between 0 and 1 (so $h = 0$ for all nodes). The g cost of a node is equal to the sum of the edges taken to reach it, and every leaf is a goal. We used the 100 random 15 puzzles first generated by Korf (1985) and the Manhattan distance heuristic. We implement beliefs numerically as discrete distributions at 100 evenly spaced values. Cserna backups use numerical integration, collapsing nearest values when necessary to keep the size at 100.

Our first experiment tests the same last incremental decision problem studied by Pemberton (1995): the LSS is the first 9 levels of a depth 10 tree. After committing to its first action, the agent observes the last level and follows an optimal path. We would expect Cserna backups to be optimal in this setting. We used 30,000 random trees. For minimin, the frontier values are the f costs, which equal the g costs. For Bellman, we estimate $\hat{f}(n)$ as $g(n) + 0.23d(n)$, where $d(n)$ is an estimate of the number of actions from n to a goal (equal to $h(n)$ in domains with unit edge costs). In trees, $d(n)$ can be calculated by subtracting the depth of node n from the depth of the tree. For Nancy and Cserna, we used both the correct beliefs, derived via Cserna backups from [0, 1] uniform distributions at the lookahead frontier, and more general (they assume nothing about the shape of the search tree nor do they assume which distribution edge costs are drawn from) approximate beliefs represented by Gaussians centered on $\hat{f}$:

$$B(n) \sim \mathcal{N}\Big(\hat{f}(n), \Big(\frac{\hat{f}(n) - f(n)}{2}\Big)^2\Big) \tag{1.2}$$

Figure 1-1 shows the mean solution cost, relative to Cserna with correct beliefs, along with

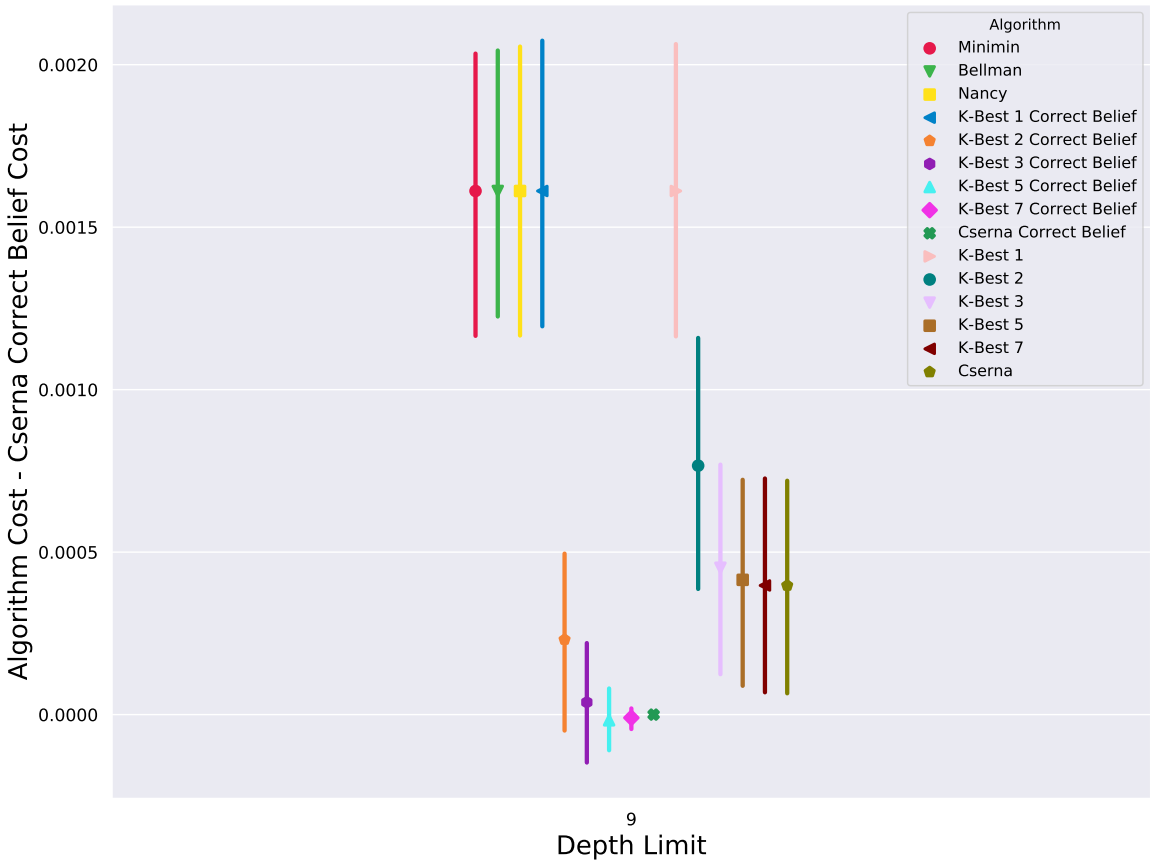

Figure 1-1: Backup rules with one level unknown.

95% confidence intervals. The top-to-bottom order in the legend corresponds to the left-to-right order in the plot. The results confirm our expectations that Cserna is optimal for the last incremental decision problem, that 1-best is equivalent to minimin, that k-best converges to Cserna as k increases, and that Nancy is equivalent to Bellman. They also indicate that Gaussian beliefs behave reasonably, leading to only a small increase in cost compared with knowing the correct belief distribution. Figure 1-2 shows the solution costs plotted as violin plots.

However, most decisions in real-time search occur far from goals. We next test the strategies in the context of limited lookahead in 1,000 trees of depth 100 (receding horizon control). We used both general Gaussian beliefs and, following Pemberton (1995), beliefs that assume only one level of the tree remains below the frontier. Because the agent can make better decisions with larger lookaheads, we used different estimates for $\hat{f}$ based on the lookahead depth or node limit, derived

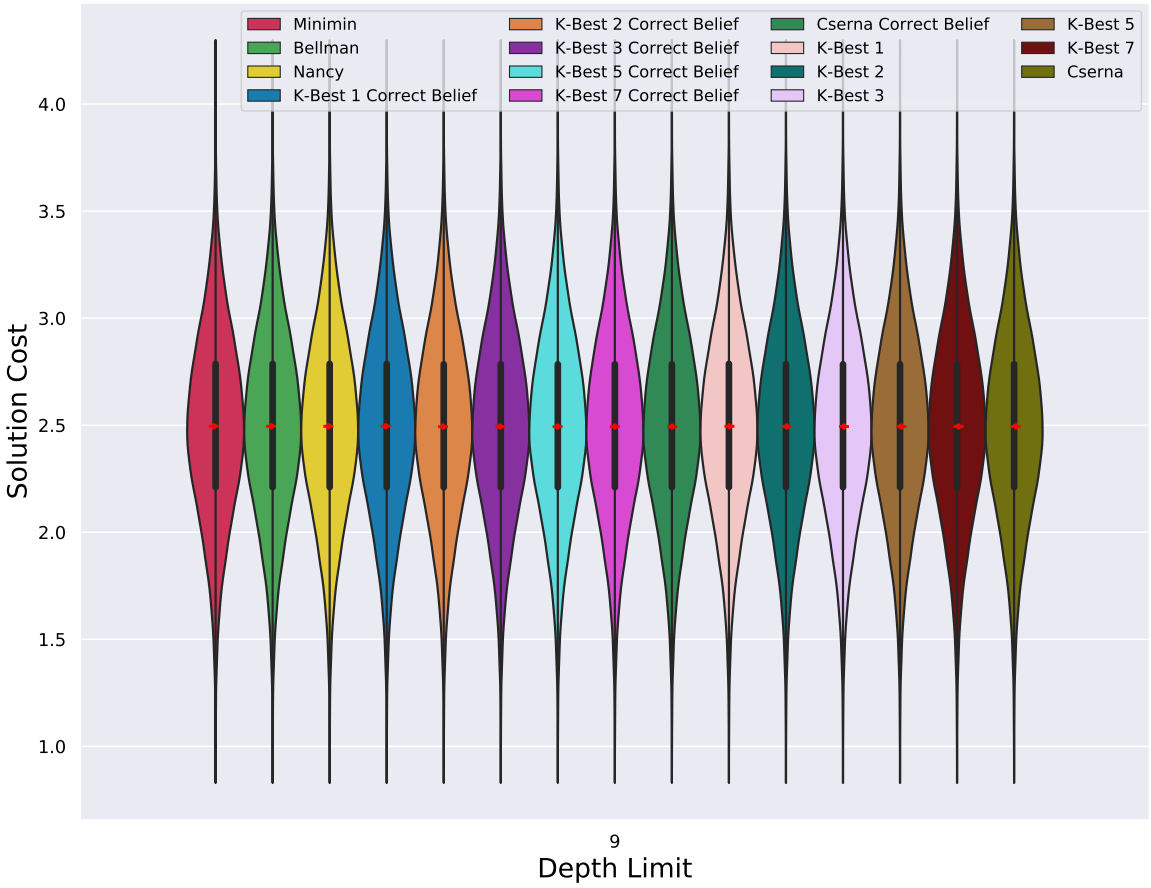

Figure 1-2: Backup rules with one level unknown. Violin plots show distribution of the data as well as a box plot with the median and quartiles. The red horizontal lines depict mean while red vertical lines show the 95% confidence intervals.

from pilot experiments on random trees ($g(n) + \epsilon \cdot d(n)$ where ϵ varied from 0.221 to 0.295). For the 15 puzzle, $\hat{f}$ was estimated using the on-line learning approach of Thayer, Dionne, and Ruml (2011) (the 'global average' one-step error model for h and d). Figure 1-3 plots relative solution cost as a function of the lookahead depth for a simple bounded depth-first lookahead. Minimin, Bellman, and Nancy perform the same, as expected. Interestingly, Cserna with Gaussian beliefs seems to perform poorly, even though the one level beliefs are no longer correct. Figure 1-4 shows the solution costs incurred by using each backup rule as violin plots.

Finally, Figure 1-5 uses A* lookahead, as used in real-time search algorithms such as LSS-LRTA*. In this case, nodes on the lookahead frontier can be at different depths. Because $h(n) =$

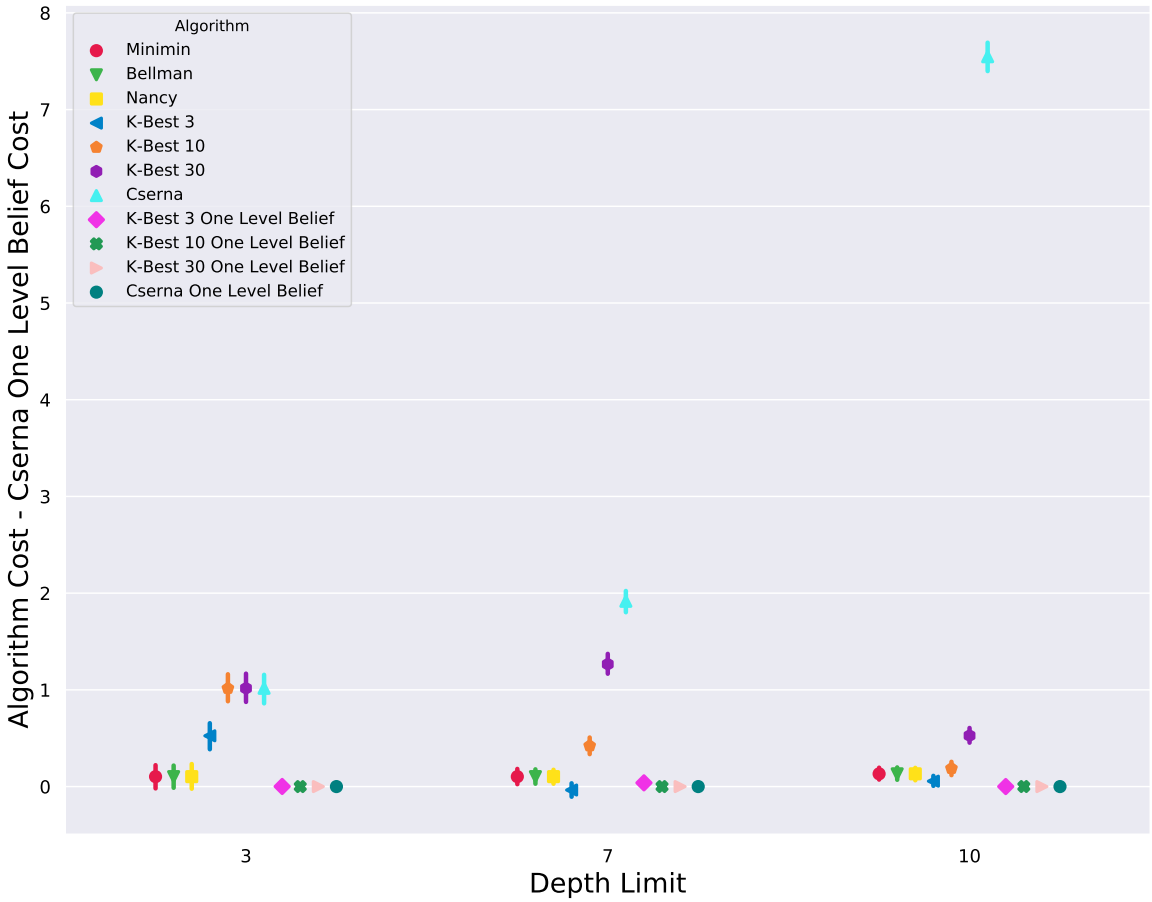

Figure 1-3: Backup rules with depth-first lookahead on trees.

0 for all nodes, minimin is essentially backing up g values and performs poorly, while Bellman and Nancy use more sensible expectations adjusted for depth and perform well. Gaussian beliefs outperform one level beliefs at lower lookaheads. Figure 1-6 shows the solution costs incurred by using each backup rule paired with A* lookahead as violin plots.

Figure 1-7 shows results on the 15 puzzle. Because this state space is a graph and the agent can return to the same state, we use LSS-LRTA* learning, which updates h values on-line and is guaranteed complete. Figure 1-8 shows the solution costs using these methods as violins. Overall, the results are remarkably similar to the random trees: Bellman and Nancy are better than minimin, and Cserna performs poorly.

We conjecture that Cserna's assumption that each child's true value will be revealed is too strong, and that real search trees are better approximated by Bellman and Nancy's assumption

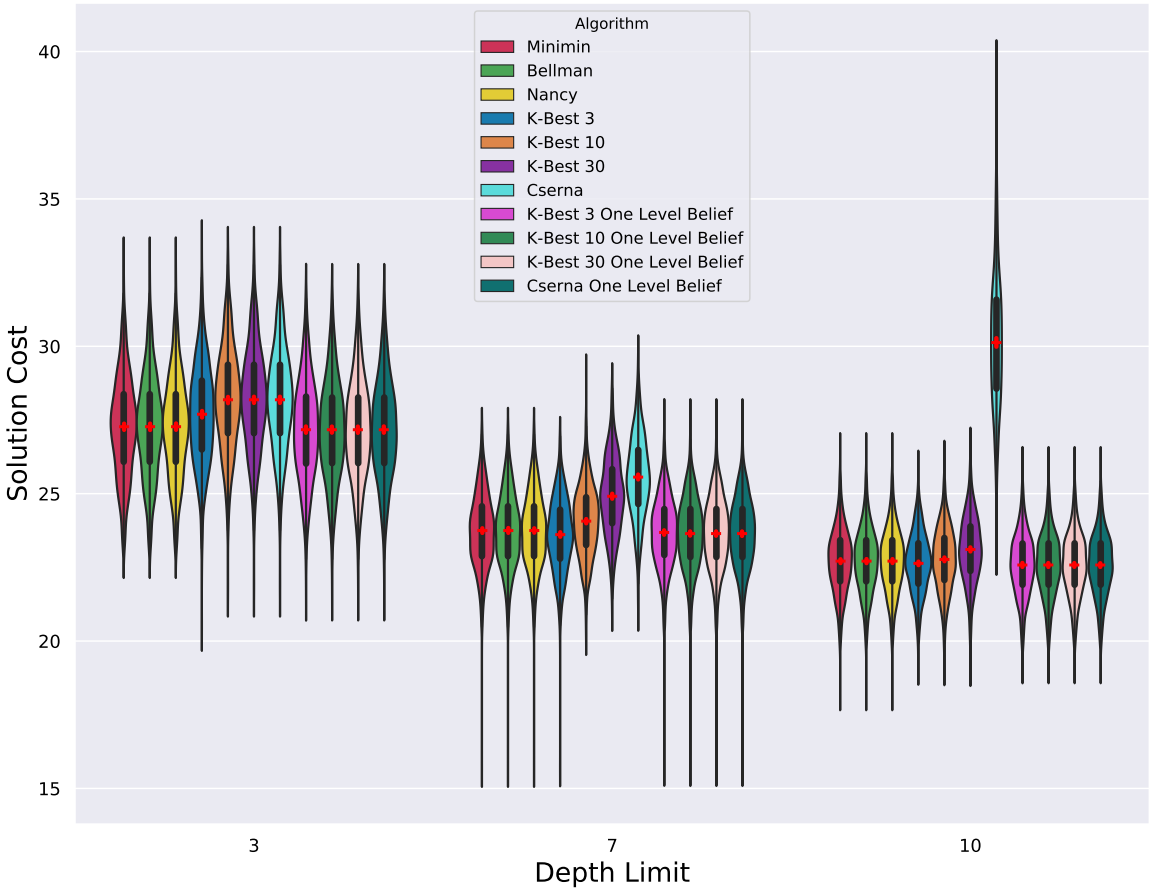

Figure 1-4: Backup rules with depth-first lookahead on trees. Violin plots show distribution of the data as well as a box plot with the median and quartiles. The red horizontal lines depict mean while red vertical lines show the 95% confidence intervals.

that little new knowledge will be brought to bear.

To summarize, we have seen that minimin and Cserna backups perform poorly for real-time search in practice, while Bellman and Nancy seem to work well. When we turn our attention to node expansion strategies, we will use Nancy backups, as they provide full belief distributions for the strategies to use.

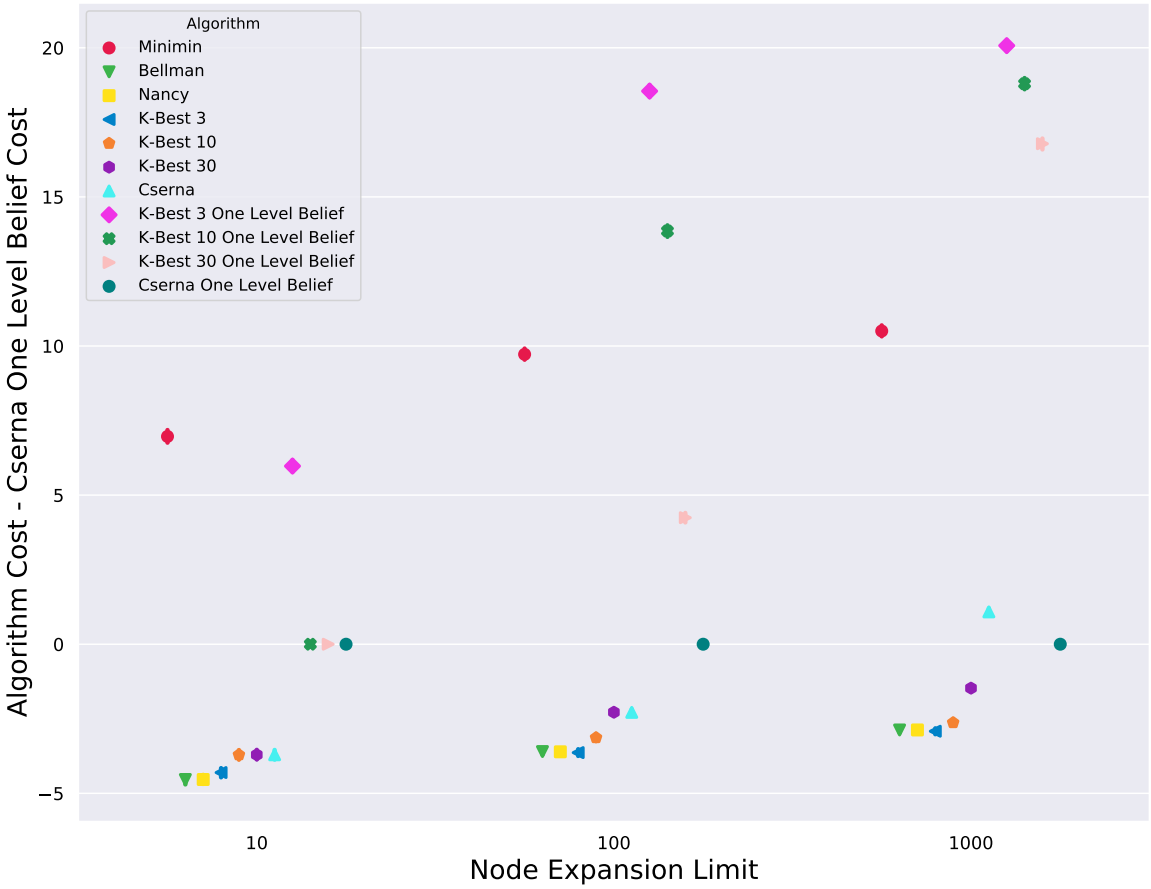

Figure 1-5: Backup rules with A* lookahead on trees.

1.3 Implementation

The following section will discuss implementation details pertaining to the backup methods, the belief model, and the random tree domain.

1.3.1 Random Tree Domain

The random tree domain can be represented as a full n-ary tree of some depth, d, where edge costs are uniformly distributed between 0 and 1. This tree is generated lazily but deterministically. Random trees are defined by three numbers: a branching factor, a maximum depth, and a random seed.

Lazy but deterministic generation ensures that while different algorithms may make different

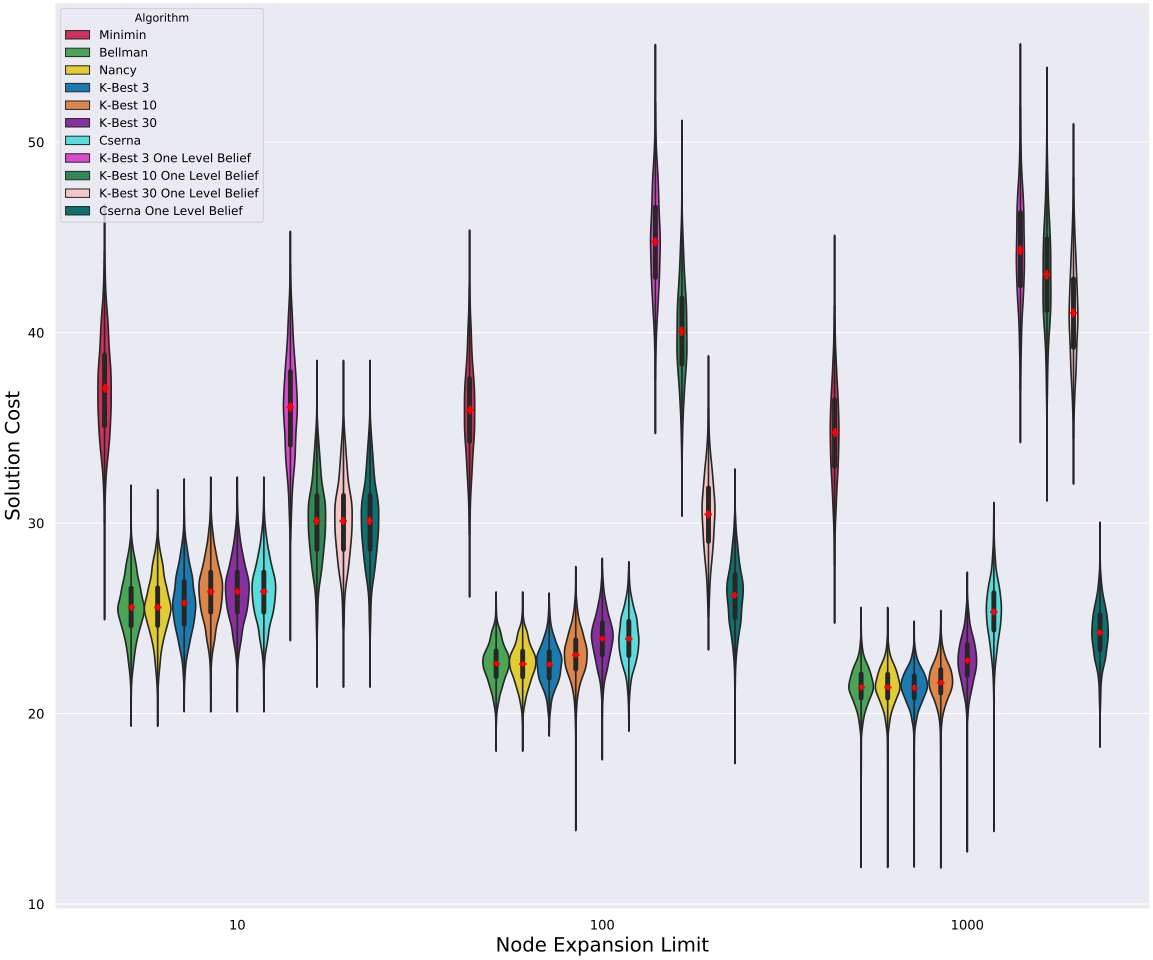

Figure 1-6: Backup rules with A* lookahead on trees. Violin plots show distribution of the data as well as a box plot with the median and quartiles. The red horizontal lines depict mean while red vertical lines show the 95% confidence intervals.

decisions, or expand different nodes, the tree instance is always the same. This can be achieved by representing states as a *depth* and an *offset* where *depth* is the depth of the node in the n-ary tree and *offset* is an integer that uniquely identifies this node. *offset* could be assigned in a variety of ways, but was implemented here as the node's order if a breadth-first traversal of the tree were to be performed, where the root has $offset = 0$. With this method of uniquely numbering states in the tree, it is trivial to find a state's successors or predecessor given its *offset*.

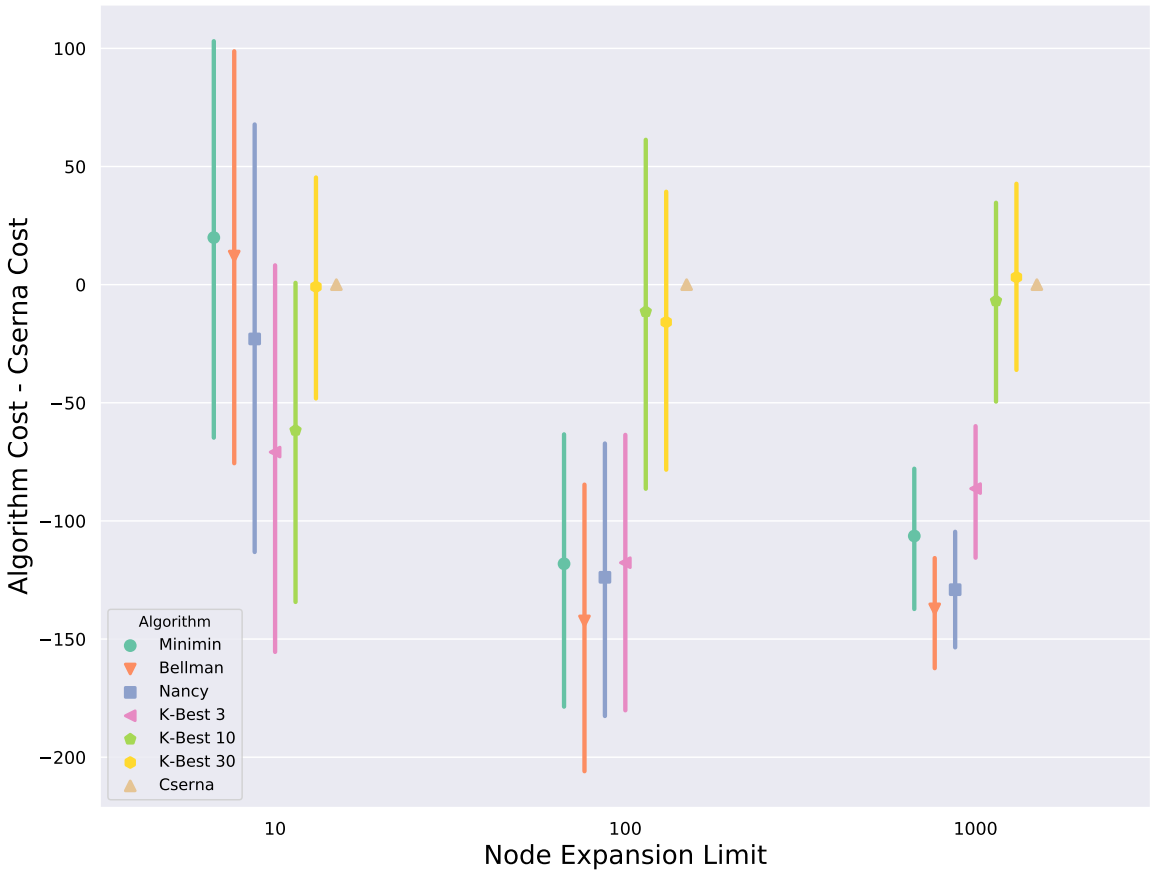

Figure 1-7: Backup rules with A* lookahead on 15 puzzles.

The *offset* of a state's predecessor is given by:

$$predcessorOffset = \frac{offset - 1}{n}$$

where n is the branching factor of the random tree. The predecessor's depth is always one less than its child's. Likewise, the *offset* values for all of a state's successors is given by:

$$successorOffset = (offset \cdot n) + i$$

where i is initially set to 1, and is incremented while it is less than or equal to the branching factor of the tree. The successor's depth is always one more than its parent's.

The offsets of the states are used in conjunction with the tree's random seed to determine which edge cost will be incurred when transitioning into the state. This is done by systematically generating and permuting 100,000 uniform random values between 0 and 1, using the random seed

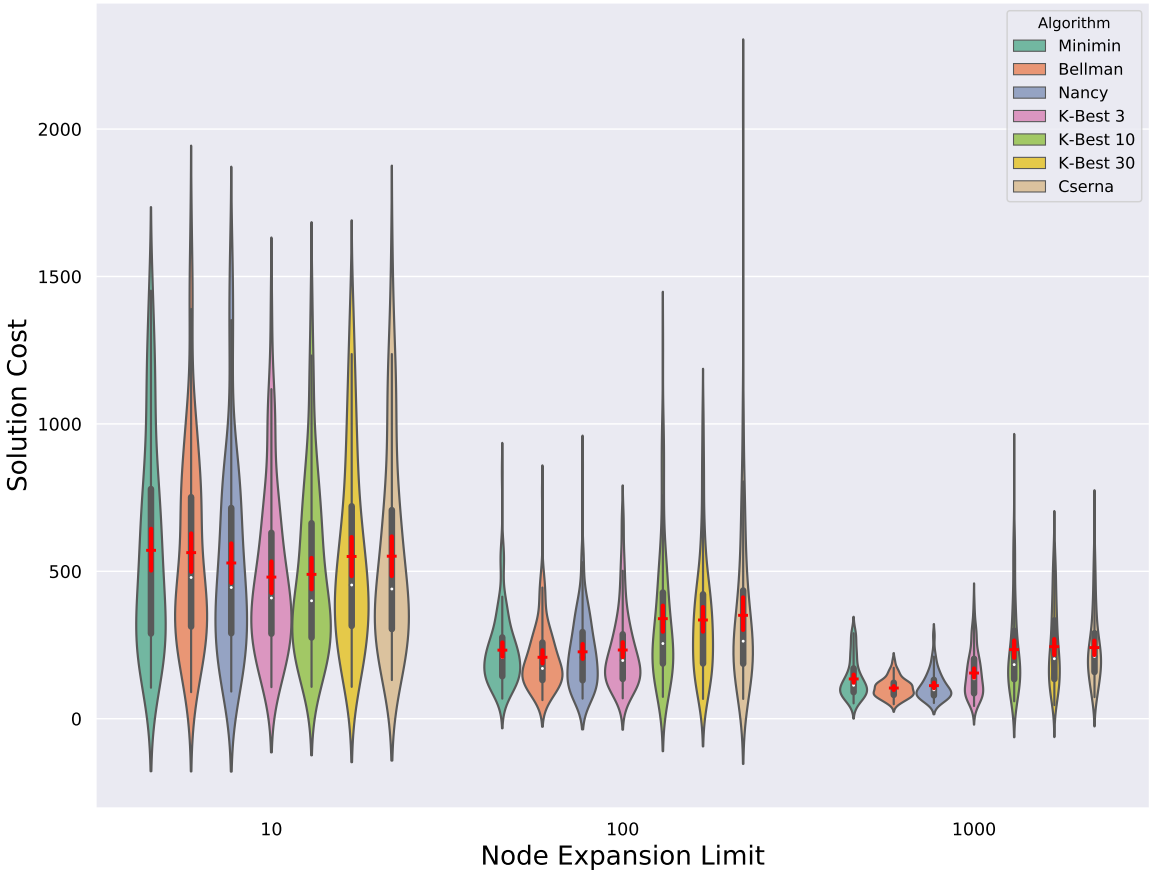

Figure 1-8: Backup rules with A* lookahead on 15 puzzles. Violin plots show distribution of the data as well as a box plot with the median and quartiles. The red horizontal lines depict mean while red vertical lines show the 95% confidence intervals.

of the tree as the random generator seed, and storing them in a lookup table when the tree instance is initialized. Edge costs are then chosen from this lookup table by modding the state's *offset* by the size of the lookup table.

$$edgeCost = lookupTable[offset \mod lookupTable.size()]$$

This method allows large random trees to be generated lazily, while ensuring that edge costs are generated deterministically regardless of the path through the tree taken by an algorithm.

1.3.2 Beliefs

Our beliefs about the true path cost through a node are represented using discrete distributions. These discrete distributions are formed from the continuous distributions obtained from Equation 1.2. The continuous distribution is truncated on the left and right.

We first truncate the left of the belief at a node's f value as this is the lowest cost that can be incurred with a path through this node, assuming admissible heuristics, and the right at three standard deviations (where one standard deviation is $\frac{\hat{f}-f}{2}$).

Finally, the truncated distribution is sampled for 100 evenly spaced values and the probabilities associated with these values are normalized so that they sum to 1. If the continuous distribution has a variance of 0, then only one sample is taken, $\hat{f}$, and that sample will have a probability of 1.

Random Tree Belief

Pemberton uses a different belief than our generalized belief based around $\hat{f}$ in (Pemberton, 1995). His belief takes into account that the edge costs of the random tree are drawn from a uniform distribution between 0 and 1. While he describes the method of obtaining these beliefs mathematically in his paper, the beliefs can also be obtained at every leaf by creating n (where n is branching factor of the tree) uniform distributions between 0 and 1, and then shifting them by the leaf's g value. These continuous uniform distributions are sampled in an identical way to the generalized beliefs based around $\hat{f}$, although these are not truncated. The n discrete distributions are then convoluted using Cserna backups to yield the belief of the leaf.

Figure 1-9 shows the belief obtained by performing the Cserna backup on n uniform distributions in a random binary tree instance. Probability Node 1 and 2 are the uniform distributions shifted by the leaf's g value, while Probability Cserna is the belief of the leaf, the result of the Cserna backup. To verify that these are the beliefs used by Pemberton, we need only to look at the cumulative distribution functions (CDF).

Figure 1-10 shows the CDF of the distributions from Figure 1-9. CDF Cserna is the CDF of the belief at the leaf. This plot matches those beliefs depicted in (Pemberton, 1995), indicting that this is a valid implementation of the beliefs used in that paper, and can be used as a means to

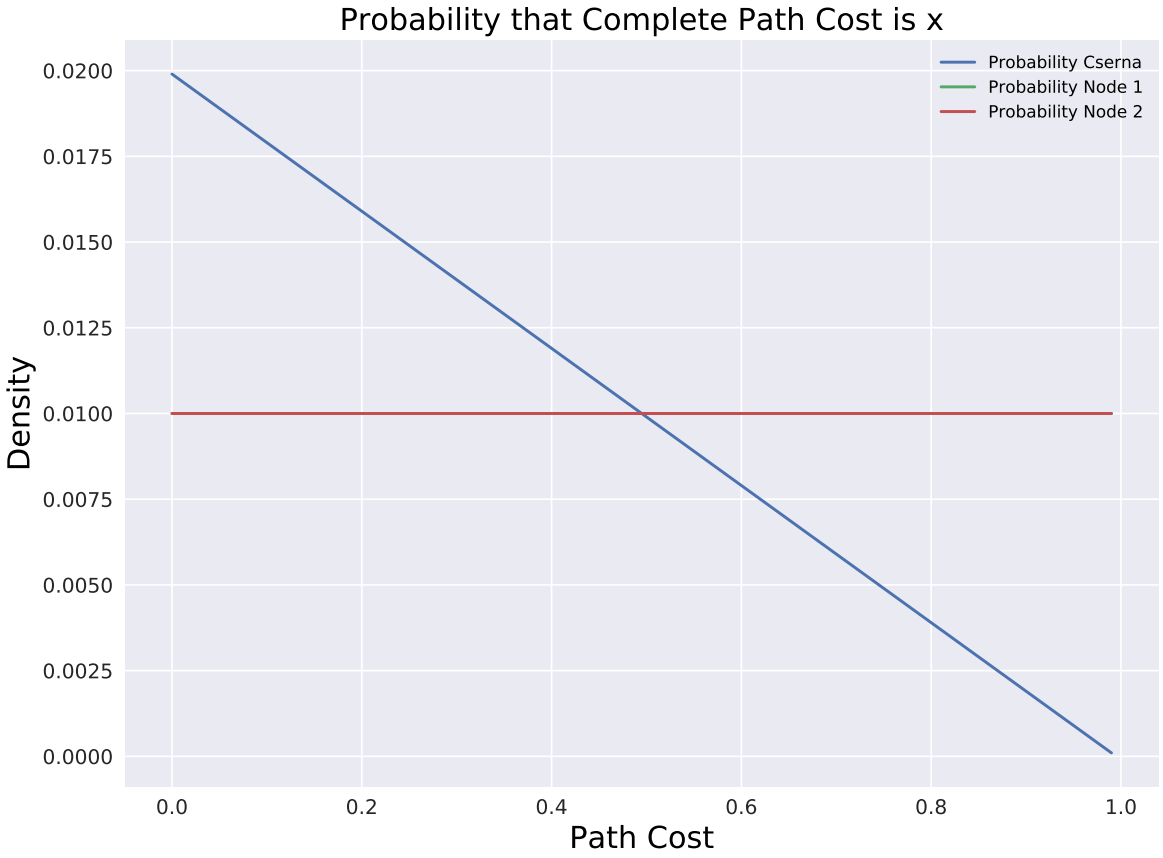

Figure 1-9: Obtaining the beliefs at the leaves as described in (Pemberton, 1995).

validate its claims about k-best.

Expected Value Operator

Belief distributions have an operator that takes the weighted sum of their samples to yield the expected value of the distribution. The expected value is used by the agent to make decisions about which TLA to execute. This value is calculated as a weighted sum rather than from a node's $\hat{f}$ value because some beliefs do not come directly from Equation 1.2, some are the result of the convolution of many beliefs, such as in a Cserna backup.

Expected values of discrete distributions that do come directly from Equation 1.2 will not be exactly equal to the mean of the original distribution, $\hat{f}$. This is due to the number of samples used, and increasing the number of samples will reduce this error at the expense of computation time. However, we assume that this error will have no effect on our decision making as all of our

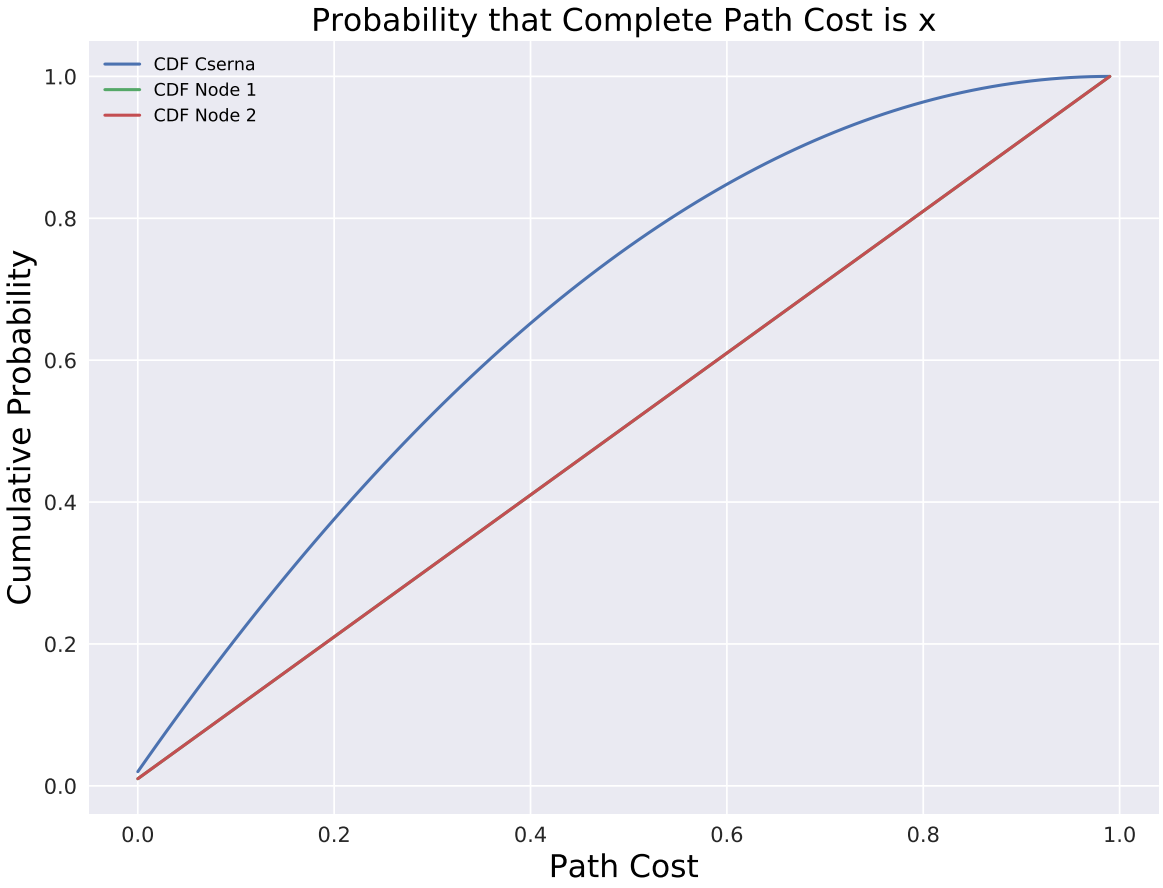

Figure 1-10: CDF of the belief at the leaves described in (Pemberton, 1995).

beliefs will use the same number of samples, and thus contain the same sampling based error when calculating the expected value of the belief.

Convolution Operator

The convolution operator is how k-best backups are applied to the discrete distributions that make up the beliefs. The operator takes two beliefs and numerically integrates over all of their samples, returning a distribution that represents the minimum potential path costs given the two available paths. The numerical integration is performed in the same way as Cserna backups are calculated in (Cserna et al., 2017)(also in Equation 1.1), except instead of returning an expected value, it returns the distribution.

The operation works by integrating over both distributions, multiplying the probabilities to-

gether and assigning that probability to the minimum of the two potential path costs in the resulting distribution. Pseudocode is given below.

Algorithm 1 k-Best Convolution

```
1: procedure CONVOLUTE(x, y)                ▷ Where x and y are discrete distributions
2:     result = ∅
3:     for sample1 ∈ x do
4:         for sample2 ∈ y do
5:             resultProb = sample1.probability * sample2.probability
6:             resultCost = min(sample1.cost, sample2.cost)
7:             result.insert(resultCost, resultProb)
```

The resulting discrete distribution should have a max sample size which is equal to the minimum size of the two distributions convoluted. If duplicate costs are encountered during integration, they are to be merged in the resulting distribution by adding their probabilities and leaving the cost unchanged.

Convoluting two discrete distributions in a fashion as described above will almost always result in the new distribution having a sample size that exceeds its maximum capacity. Therefore, there must be a way to merge neighboring samples in a distribution, so as to shrink the sample size. When the number of samples in a discrete distribution exceeds its maximum sample size, the distribution will need to be resized. This is done by repeatedly pairing up adjacent samples, computing the distance between them, and merging the closest samples until the desired sample size is reached.

Samples are ordered according to their cost, and adjacent samples in the resulting list are grouped into pairs. These pairs are then sorted based on their distance, which is just the difference between the costs of the samples in a pair. Finally, pairs are merged, starting with those with the smallest distance. Merging a pair means that two samples are removed from the distribution, and a new one is added. The new sample has a probability that is the sum of the two probabilities from the pair's samples. The new sample's cost is the sum of the paired sample's costs, weighted by the

ratio of their probabilities to the new sample's probability.

The new probability is given by:

$$newProbability = P(pair_{sample1}) + P(pair_{sample2})$$

The new cost is given by:

$$newCost =$$
$$\frac{P(pair_{sample1})}{newProbability} \cdot Cost(pair_{sample1}) +$$
$$\frac{P(pair_{sample2})}{newProbability} \cdot Cost(pair_{sample2})$$

When a pair is merged, any other pairs containing either of the merged samples must be updated to now include the new sample from the merge. This process is repeated until the number of samples no longer exceeds the distribution's maximum sample size.

The convolution operator can be applied iteratively on pairs of beliefs if $k > 2$. Convolution order is not relevant as this operation is associative.

Figure 1-11 shows the CDF of the belief that is obtained after using the convolution operator on two leaf nodes. CDF Node 1 and 2 are the CDFs of the beliefs at the leaf nodes, using the beliefs described in (Pemberton, 1995). CDF Cserna is the CDF of the belief obtained by the convolution, which in this case is the Cserna backup of two frontier nodes in a binary tree. This plot is similar to Figure 3 in (Pemberton, 1995), which indicates that the convolution operator correctly performs k-best backups, and that k-best is just a generalization of Cserna backups.

Figure 1-12 shows the operator being used to perform a Cserna backup of two frontier nodes where the belief is our generalized belief based around $\hat{f}$. Probability Node 1 and 2 are the beliefs at the leaf nodes, and Probability Cserna is the belief obtained by the Cserna backup.

Squish Operator

To guide expansion, we will need to model how additional exploration under a TLA will change our belief. We make the assumption that additional search will change our beliefs by shrinking the variance and leaving the mean unchanged. This is because the variance is determined by heuristic error and heuristic error is assumed to decrease as the frontier gets closer to the goal, which is a

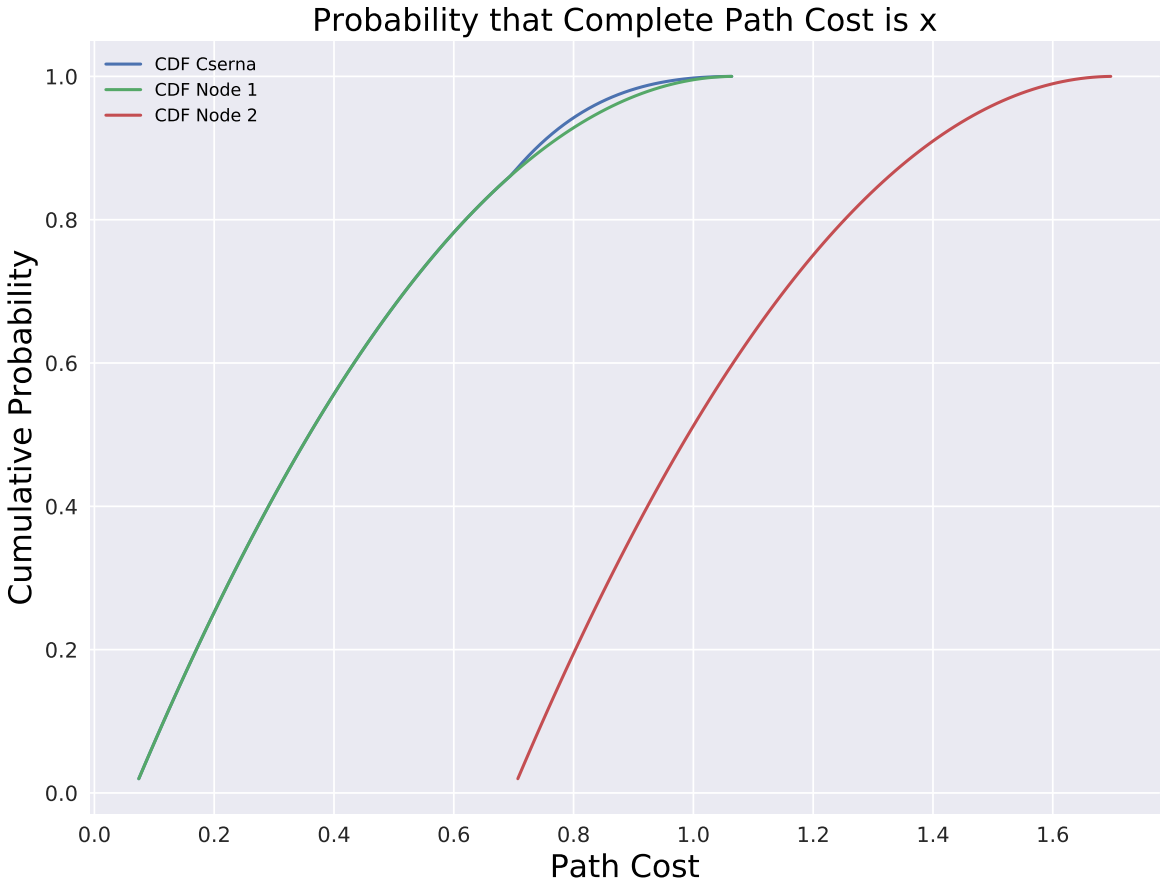

Figure 1-11: CDF of the belief that is obtained from the Cserna backup of two nodes using beliefs from (Pemberton, 1995).

result of exploring deeper into a search space. The squish operator takes a discrete distribution and a squish factor, and moves all of the costs in the distribution towards the mean according to the squish factor.

The squish factor can be viewed as what percentage of the way the expansion will bring the agent to the goal. It is defined as

$$squishFactor = \min(1, \frac{ds}{dy})$$

where ds is defined as

$$ds = \frac{1}{averageExpansionDelay}$$

and dy is the d value (distance heuristic, or the number of actions a node is from the goal) of the

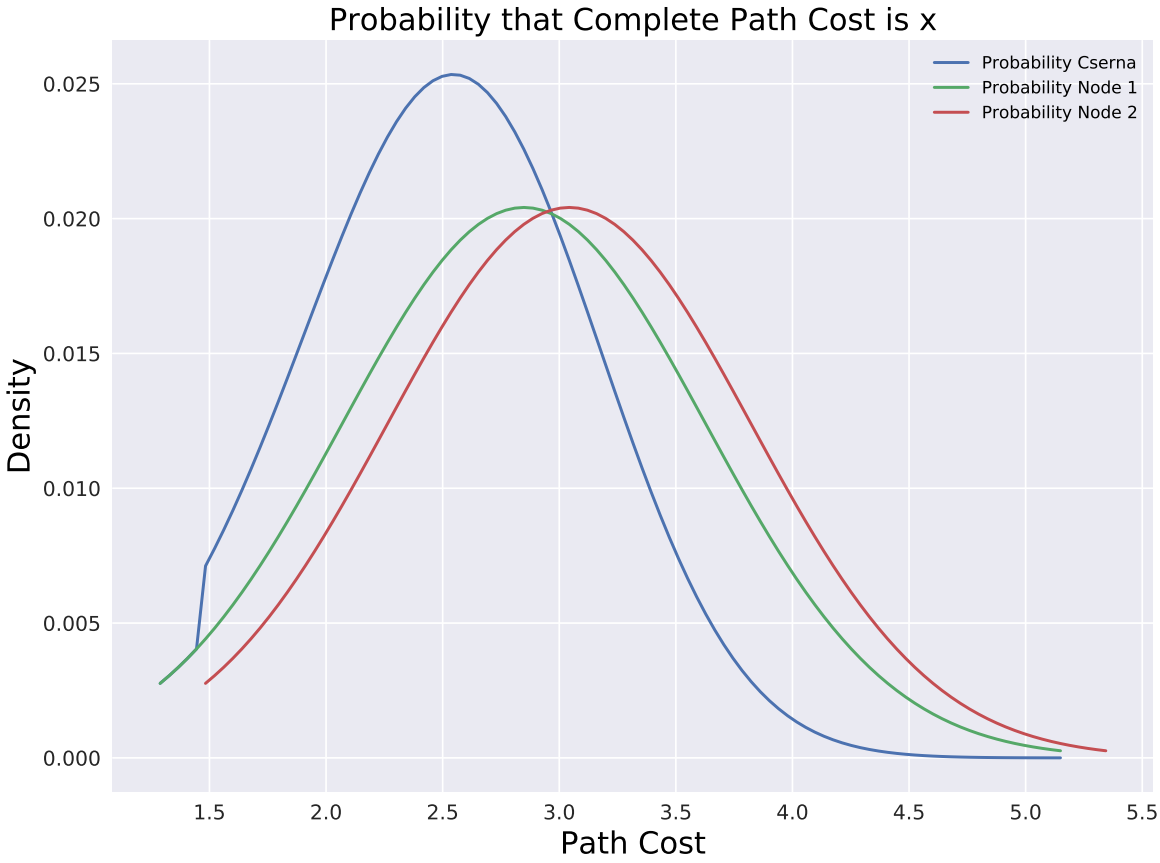

Figure 1-12: Belief obtained from performing Cserna backup on two nodes using generalized beliefs based around $\hat{f}$.

best node under a TLA. Expansion delay is the number of expansions between when a node is generated and when it is finally selected for expansion. This average can be tracked using a sliding window that averages the expansion delays of the last 100 nodes expanded.

If more than one expansion is going to be allocated under a TLA in a single iteration, then ds can be defined more broadly as:

$$ds = \frac{n}{averageExpansionDelay}$$

Where n is the number of expansions that will be allocated under the TLA during this iteration. Expansion delay can also be viewed as how many expansions it takes before the search makes progress towards the goal on a single path that it is currently exploring (as search can jump between exploring multiple different paths from iteration to iteration). Given this definition of

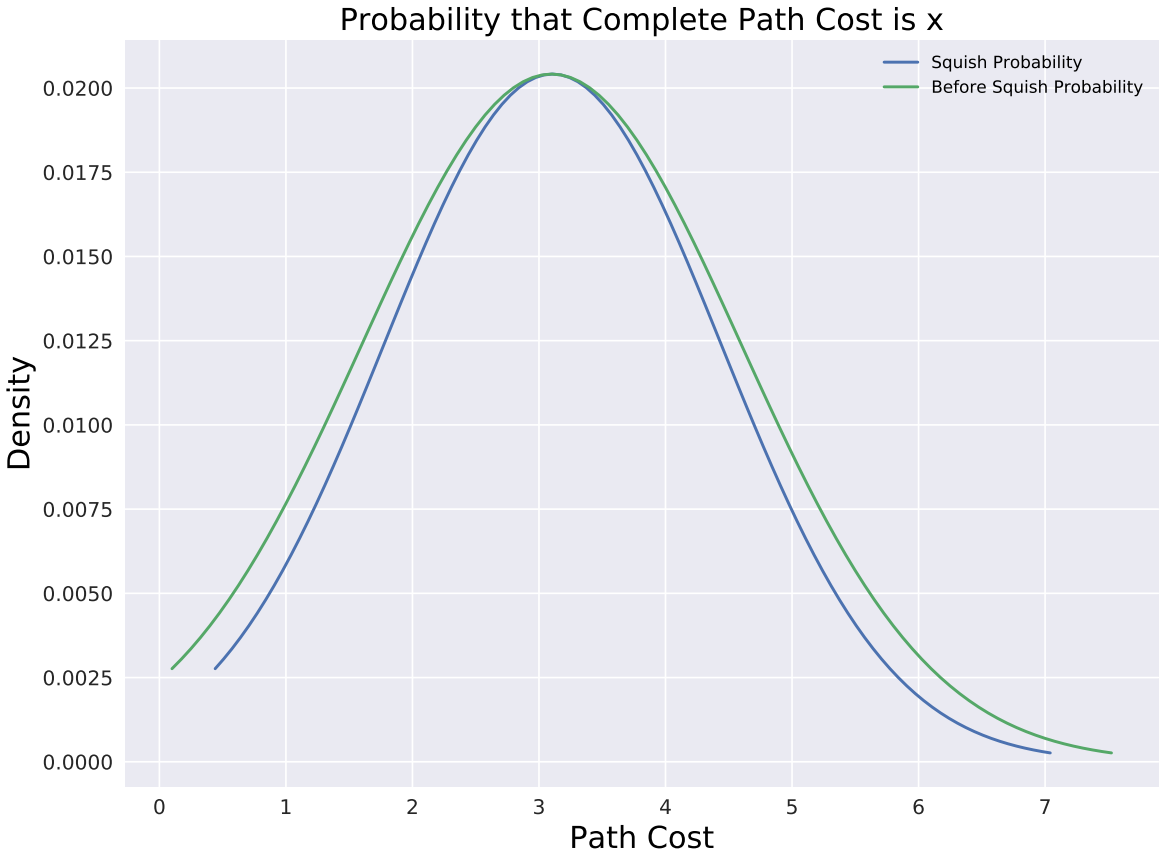

Figure 1-13: Belief that has been squished to estimate how additional search will effect the belief at the TLA.

expansion delay, *ds* can be viewed as the ratio of allocated expansions to the number of expansions needed to make progress towards the goal, and is therefore an estimate of how far the allocated expansions will actually progress towards the goal. Dividing this by how much progress is actually needed to reach the goal (the distance to go estimate from the state being expanded, *dy*) yields the squish factor, which can be 1 (search progresses all the way to the goal) or some number less than 1 and greater than 0 (search makes progress towards the goal, but does not reach it).

Once this squish factor is obtained, it can be used to update all of the costs in the discrete distribution. If the factor is equal to 1, the belief will be a spike at the mean with probability of 1. Otherwise the distance from all costs to the mean cost are calculated. The costs in the distribution are then shifted towards the mean by their distance from the mean, times the squish factor. Figure 1-13 shows the result of performing a squish on the belief of a TLA.

1.3.3 Calculating ϵ for $\hat{f}$

The value of ϵ used to calculate $\hat{f}$ in the experiments (where $\hat{f}(n) = g(n) + h(n) + \epsilon \cdot d(n)$) was determined based on the domain used, the expansion algorithm used, and the node expansion/depth limit or by calculating the 'global average' one-step error.

In the random trees domain, ϵ values were pulled from pilot experiments using depth-first and A* expansion strategies. The lowest average solution costs for each expansion strategy-expansion limit pair were taken from these results (shown in Figures 1-14 and 1-15) and divided by the depth of the tree to get an ϵ value to be used for that expansion strategy and node expansion limit. If additional ϵ values were needed in later experiments, they were obtained by interpolation from the pilot experiments. The results of these pilot experiments are shown in Figures 1-14 and 1-15.

Table 1-1 shows the ϵ values used in the experiments for random trees at different depth expansion limits when depth-first expansions were used. Likewise, Table 1-2 shows the ϵ values used for various node expansion limits when A* expansion was used. The values from Table 1-2 were also used for the expansion strategies tested in chapter 2 when they were run on random trees.

Table 1-1: Random Tree ϵ Values With Depth-First Expansion

Lookahead Depth Limit	ϵ **Value**
3	0.27
7	0.24
9	0.23
10	0.225

When calculating ϵ for the sliding tile puzzle, 'global average' one-step error was used in place of values drawn from pilot experiments. This is done the same way as explained by Thayer et al. (2011).

In the random tree domain, the true distance heuristic, $d^*(n)$, was able to be computed using the depth of the state and the maximum depth of the random tree and can be used as $d(n)$ in the

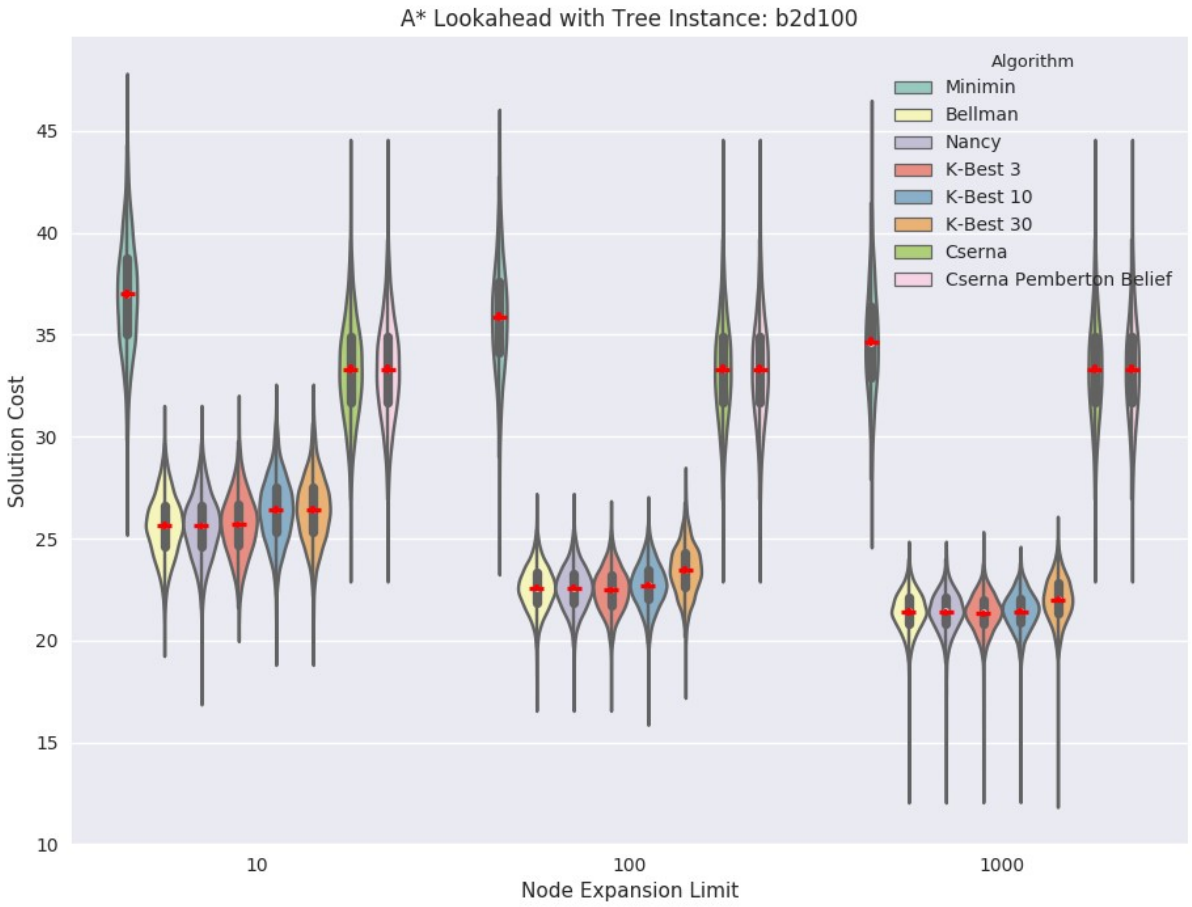

Figure 1-14: Pilot experiment results on random binary tree of depth 100 with A* expansion.

$\hat{f}$ calculation. However, the sliding tile puzzle uses the Manhattan distance heuristic for both $h(n)$ and $d(n)$. The Manhattan distance tends to underestimate the true distance by about a fifth, and therefore we not only have to correct $h(n)$ for our $\hat{f}$ value, but also $d(n)$. This means that we have to obtain two ϵ values: ϵ_h for $h(n)$ and ϵ_d for $d(n)$.

The 'global average' one-step error model assumes that the heuristic error is equally spread across all edges in a graph. It is calculated when expanding nodes, taking the difference between the parent node and its best child (lowest f value) of either $f(n)$ for ϵ_h or $d(n)$ for ϵ_d. These differences can be added to a running sum and divided by the number of expansions used in the

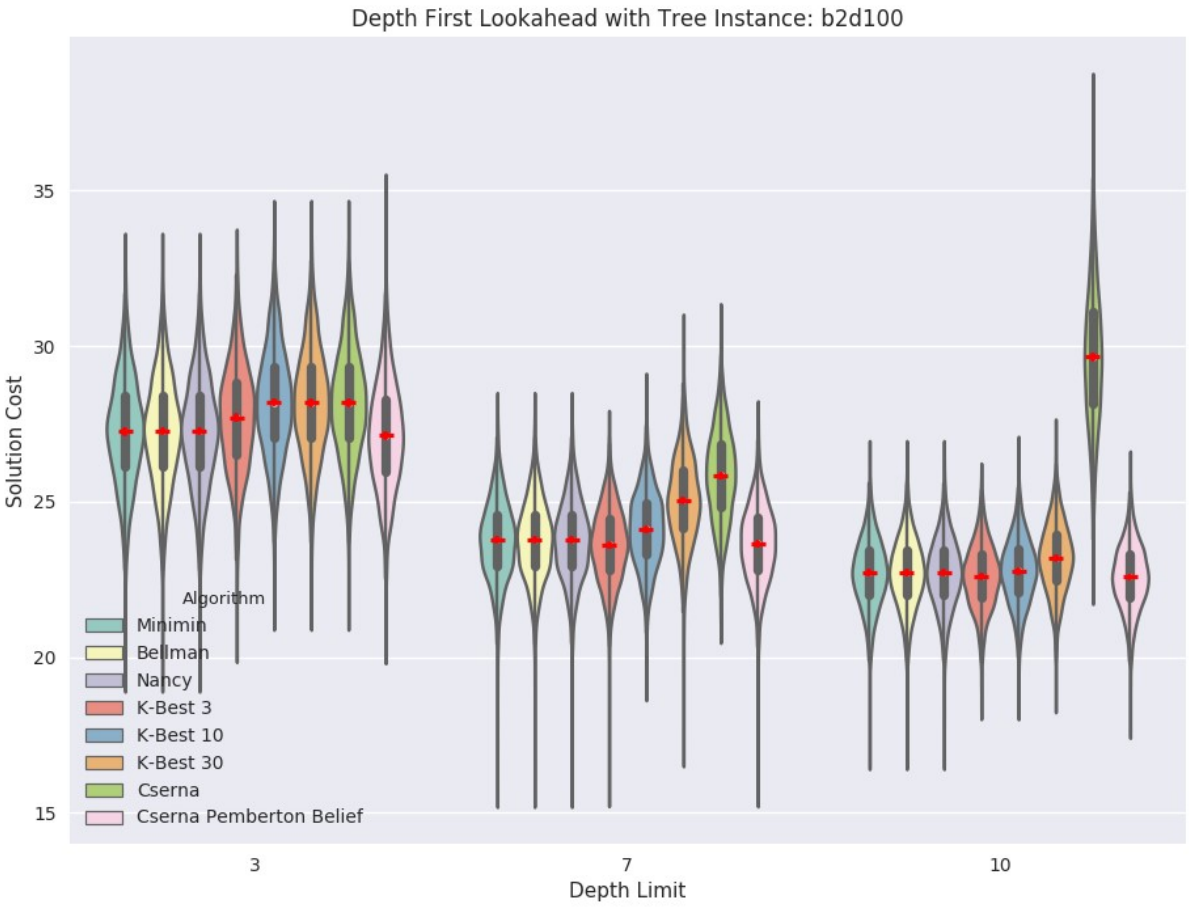

Figure 1-15: Pilot experiment results on random binary tree of depth 100 with depth-first expansion.

sum to yield the average errors $\bar{\epsilon_h}$ and $\bar{\epsilon_d}$.

$\bar{\epsilon_h}$ and $\bar{\epsilon_d}$ are used to correct heuristic error in both $h(n)$ and $d(n)$ in an attempt to better approximate the true heuristic values, $h^*(n)$ and $d^*(n)$. The corrected distance heuristic, $\hat{d}(n)$, is given by:

$$\hat{d}(n) = \frac{d(n)}{1 - \bar{\epsilon_d}}$$

The corrected heuristic, $\hat{h}(n)$, is given by:

Table 1-2: Random Tree ϵ Values With A* Expansion

Node Expansion Limit	ϵ **Value**
3	0.295
10	0.26
30	0.23
100	0.225
300	0.223
1000	0.221

$$\hat{h}(n) = h(n) + \hat{d}(n) * \bar{\epsilon_h}$$

These calculations are slightly more complicated when heuristic learning is involved, such as in LSS-LRTA*. This is because in the heuristic learning step of LSS-LRTA*, h values are propagated back from the frontier as they are closer to the goal and thus more accurate. This means heuristic error is only present over the distance between the frontier node and the goal. As a result, the d values of the interior nodes have to be updated accordingly, as this value is used in conjunction with $\bar{\epsilon_h}$ to correct for heuristic error incurred at each hop. However, as the heuristic value of the interior node was passed back from its successor on the frontier, the distance over which this error is incurred is no longer equal to that node's d value, it is equal to the d value of the successor it inherited h from.

For example, if y is a node on the frontier and x is a node that receives an h value from y, then $\hat{f}(x)$ is given by:

$$\hat{f}(x) = g(x) + h(y) + \frac{d(y)}{1 - \bar{\epsilon_d}} * \bar{\epsilon_h}$$

So x would receive the d value of whichever frontier node it receives h from (a similar example is given by Kiesel et al. (2015) in Figure 6). However, this cannot be done by simply overwriting $d(x)$ because the full distance between x and the goal is still needed for calculating $\bar{\epsilon_d}$ and should

also be learned in the learning step as, like h, it is calculated using the Manhattan distance and contains error.

Therefore, we differentiate between the distance between a node and the goal, $d(n)$, and the distance over which its heuristic error is incurred, $derr(n)$. Going back to the example with nodes x and y, $\hat{f}(x)$ is now calculated with:

$$\hat{f}(x) = g(x) + h(y) + \frac{derr(y)}{1 - \bar{\epsilon_d}} * \bar{\epsilon_h}$$

In this example, $derr(x) = derr(y) = d(y)$. In general, $derr(n) = d(n)$ when a state is generated for the first time, however after the first learning step this may no longer be the case, as a node may receive an h value from one of its successors (and thus receive that successor's $derr$ value as well).

As $d(x)$ is now preserved, it can be corrected in the same way as $h(x)$ is in the learning step, and used to calculate $\bar{\epsilon_d}$ in later iterations of the search. It is very important that $derr(x)$ is not used to calculate $\bar{\epsilon_d}$, as that will not yield correct one-step error values. This also means that $\hat{d}(n)$ would be updated to be:

$$\hat{d}(n) = \frac{derr(n)}{1 - \bar{\epsilon_d}}$$

Thus, in general, $\hat{f}$ is given by:

$$\hat{f}(x) = g(x) + \hat{h}(x)$$

$$\hat{f}(x) = g(x) + h(x) + \hat{d}(x) * \bar{\epsilon_h}$$

$$\hat{f}(x) = g(x) + h(x) + \frac{derr(x)}{1 - \bar{\epsilon_d}} * \bar{\epsilon_h}$$

More information on the 'global average' one-step error model is given by Thayer et al. (2011) and Thayer (2012).

1.3.4 Top Level Actions

Top level actions are the actions that are immediately available to an agent from its current state. As the TLAs are needed to guide the rest of the search, the first expansion of a search iteration is responsible for initializing the objects that represent the TLAs.

TLA Composition

TLAs were modeled as objects with a belief, a node pointer, and a heap. The belief of a TLA represents the agent's belief about the true path cost through that action. We defined this belief about f^* as $f^{@}$ in the section on Bellman backups (see section 1.1.2). This belief is pulled up to the TLA from the frontier in a manner that is dictated by the backup rule being used. We define the best TLA, α, as the one with the lowest expected cost, which can be obtained by taking the expected value of its belief distribution.

The node pointer in a TLA points to the immediate successor of the current state that is accessible via this TLA. As we are using one-step commitment, this is the state that the agent will transition to and begin the next search iteration at if this TLA is selected for execution.

Finally, the heap of the TLA contains all of the nodes on the open list beneath this action. The heap is a min-heap sorted by $\hat{f}$, the expected cost through a node. This heap is used by Nancy, Cserna, and k-best backups to find the best k nodes under a TLA where these nodes are defined as those with the lowest $\hat{f}$ values. The heap is also used to identify which node (the one with lowest $\hat{f}$) would be expanded under a TLA in risk-based expansion, and is central to calculating the risk metric.

These heaps could be used either alongside or completely in place of a traditional open list when using minimin or Bellman backups, or any traditional expansion strategy (A*, $\hat{f}$, breadth-first) depending on how they are sorted. Assuming there are n TLAs, the traditional open list can be obtained through the union of the heaps of all the TLAs.

$$open = heap_{TLA_1} \cup heap_{TLA_2} \cup ... \cup heap_{TLA_n}$$

For example, the node to expand in A* would be found by iterating over the TLAs and selecting

the top heap node which minimizes f (assuming all heaps are sorted by f). Minimin backups could similarly be performed by backing up the belief of the node on the top of the heap to each TLA, and then executing the TLA with the lowest sample in its belief (as beliefs are truncated on the left at the node's f value).

Pruning Duplicate Nodes

Duplicate nodes were pruned from under the inferior TLA (the TLA under which the state appears with higher g). This process is similar to how LSS-LRTA* rewires parent pointers of nodes to point to the parent through which the newly generated node has the lowest g value, except here we are also removing the node from the inferior TLA's heap and placing it into the heap of the superior TLA. This is the same behavior as algorithms that use a single open list and rewire parent pointers.

In trees, this pruning does not impact performance of Nancy, Cserna, or k-best backups. However, in graphs (where this pruning will actually take place) it will essentially prune these sub-optimal paths from the TLA, and they will not be reflected on the backup as the pruned node will be removed from the top k in the heap. This pruning therefore leads to an approximation of the Cserna, Nancy, or k-best backups, as the true backups would require duplicate states under different TLAs to be stored in different nodes and tracked separately. Instead of performing this perhaps costly duplicate node tracking, we opted to prune.

CHAPTER 2

Choosing Nodes to Expand

Given an agent's current beliefs about the $f^{@}$ values of the TLAs, it is not immediately obvious which frontier node to expand.

Mutchler (1986) showed that the most common choice, the node with lowest f, is not optimal. Kiesel et al. (2015) propose $\hat{f}$, but as Figure 0-2 showed, taking our uncertainty into account is also important. Because we are using Nancy backups, the only node whose expansion can change our belief about a TLA is the one with the minimum $\hat{f}$. We will consider several candidate criteria to use for choosing which node to expand, and we optimize them myopically. We predict, for a given TLA, how our belief about it would change if we were to expand the best frontier node under it. (We discuss one way of forming these predictions below.) We then evaluate the resulting belief in the context of the other TLAs' unchanged beliefs according to a desirability criterion, and choose the node whose expansion gives the most desirable set of beliefs. The criteria we consider are confidence, expected advantage, and risk. Because action selection will choose the TLA with the lowest expected value, we will call the current such TLA the 'best action' or α.

2.1 Theory

We define confidence as the probability, given a set of beliefs, that the best action actually has the lowest $f^{@}$ value. Said another way, this is the probability that, given samples from each TLA's belief, the one from α is lowest. By expanding nodes so as to maximize our confidence, we reduce the overlap among the leading beliefs, helping us be sure that we choose the best action. In Figure 0-2, for example, expanding under β may well increase confidence faster than expanding under α. While this seems sensible, note that confidence can have odd behavior. For example, an action β might have most of its belief's probability mass below the belief for α, making β very likely to be lower,

yet an extreme outlier in β's distribution could cause its $\hat{f}$ to be higher than α's. The central weakness of confidence is that, as a probability, it does not take solution cost into account. Given two possible expansions that equally improve the probability that α is best, it cannot distinguish which might improve the agent's expected cost more.

To capture this, we define expected advantage, given a set of beliefs, as the difference between the expected costs of the best action $\hat{f}(\alpha)$ and the second best action $\hat{f}(\beta)$. Note that expanding nodes to maximize expected advantage is not the same as expanding the node with lowest $\hat{f}$ — it might be the case that expanding under β moves it further from α than expanding under α moves it away from β. There are two serious deficiencies in this criterion, however. First, it does not take into account the uncertainty of our beliefs and cannot distinguish between two expansions that move $\hat{f}$ values equally but reduce variance differently. Second, and related to this, in practice we will model belief change due to search as reduction in variance, leaving $\hat{f}$ unchanged. So a criterion insensitive to uncertainty is a non-starter.

Finally, we arrive at our final criterion, risk, which we define as the expected regret in those cases in which α was not the correct choice. For two TLAs, this is

$$\mathbb{E}\left[f^{@}(\alpha) - f^{@}(\beta) \mid f^{@}(\beta) < f^{@}(\alpha)\right],$$

where $\mathbb{E}$ is the (conditional) expectation operator. This generalizes to additional TLAs by considering every outcome where one of the β_i is better than the current α. This is done with the TLAs' current beliefs in concert with estimates about how each belief would change if we were to search under its corresponding best frontier node (done by squishing the belief of the TLA we are currently simulating expansion under). Numerically, risk is computed by taking each combination of possible values for the best TLA, α, and the other TLAs, β_i, and finding $a - b_i$ where a is a possible $f^{@}$ value in α and b_i is a possible $f^{@}$ value in β_i where $b_i < a$, weighted by the probability of that combination. We do this computation once for each TLA, temporarily adjusting the belief of that TLA to be the predicted post-expansion belief. The predicted post-expansion belief about a TLA has the same mean as the pre-expansion belief, but a smaller variance because variance is a function of heuristic error, which we assume will decrease as additional expansions bring the search closer to a goal. We then expand the frontier node under the TLA whose predicted post-expansion

belief resulted in the least risk.

Risk combines the attractive properties of confidence and advantage, being sensitive both to the probability that α is the better decision and the regret we will experience when it is not. We will call real-time search using Nancy backups and risk-based expansion Nancy for short.

2.2 Experiments

We now turn to an experimental evaluation of node expansion strategies. To use risk-based expansion, we require a model of how beliefs change due to search. We use a variant of the scheme presented by Cserna et al. (2017). Note that the variance of our beliefs are based on the distance to the goal from the frontier node from which we inherited our belief ($\hat{f} - f$, Eq. 1.2). Expanding the node will reduce our distance to goal, thus we reduce its variance by $\min(1, d_s/d(n))$ (Eq. 2 of Cserna et al. (2017)), where $d(n)$ is the estimated number of steps to the goal and d_s is the *expansion delay* (a measure of search vacillation equal to the average number of other nodes generated between when a node is generated and when it is expanded). We assume the expected value remains unchanged (its expected behavior).

We test expansion strategies in the context of Nancy backups, given their success in the experiments above. Frontier nodes are given Gaussian beliefs (Equation 1.2). We compare risk-based expansion with conventional lookahead techniques, including breadth-first, A*, and $\hat{f}$-based expansion. Figure 2-1 shows mean solution cost relative to A* on random trees of depth 100. We see that, for lookaheads of more than 3 nodes, breadth-first is dominated by the more flexible strategies. $\hat{f}$ lookahead is often better than A*'s f-based strategy, confirming the results of Kiesel et al. (2015). But risk-based expansion is clearly the best, showing strong benefits over the other strategies for lookaheads of 10 nodes and greater. Figure 2-2 shows the solutions costs incurred using these methods as violin plots.

Figure 2-3 presents results from 15 puzzles. Breadth-first is again poor, but $\hat{f}$ is not clearly superior to f. However, aside from poor performance with a 3-node lookahead, risk-guided expansion again seems superior. For comparison, we also show the performance of LSS-LRTA*, a popular real-time search method that uses A* expansion, minimin backups, and moves toward the

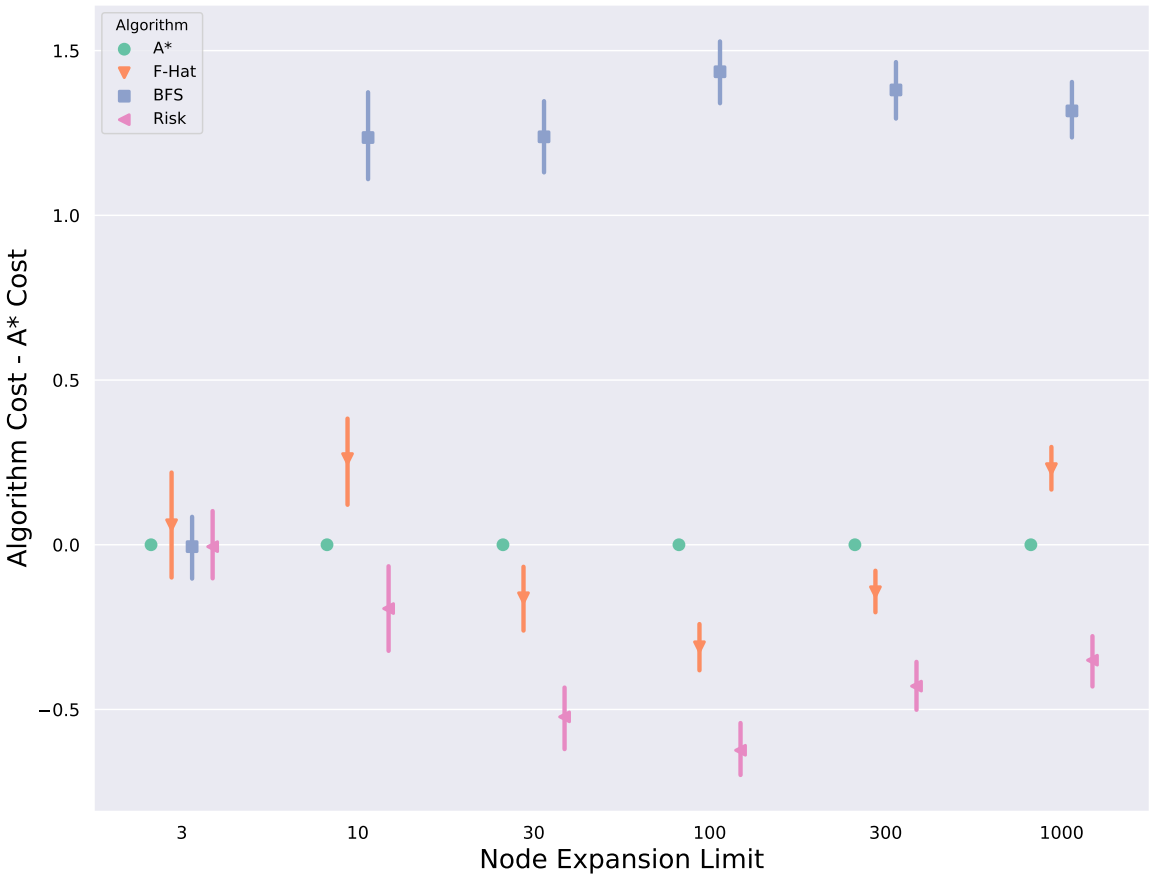

Figure 2-1: Expansion strategies on random trees.

frontier node of lowest f. The Nancy algorithm of risk-based expansion, Nancy backups, and $\hat{f}$ action selection offers significantly better performance at lookaheads of 30 and higher. Although it may appear that risk is converging back toward A* as lookahead increases, this is because both algorithms are approaching optimal solutions. The viloin plots for these solution costs are depicted in Figure 2-4. Figure 2-5 shows the average optimality gap (cost over optimal) of each method as a percentage of A*'s average gap. The results show that risk has 65% of A*'s difference from optimal at a lookahead of 1000, and even less at 300.

To summarize, we have seen that a lookahead strategy based on minimizing risk can outperform traditional search strategies. By using the belief distributions provided by Nancy backups, it can determine when it may be beneficial to expand nodes other than those having the best expected value.

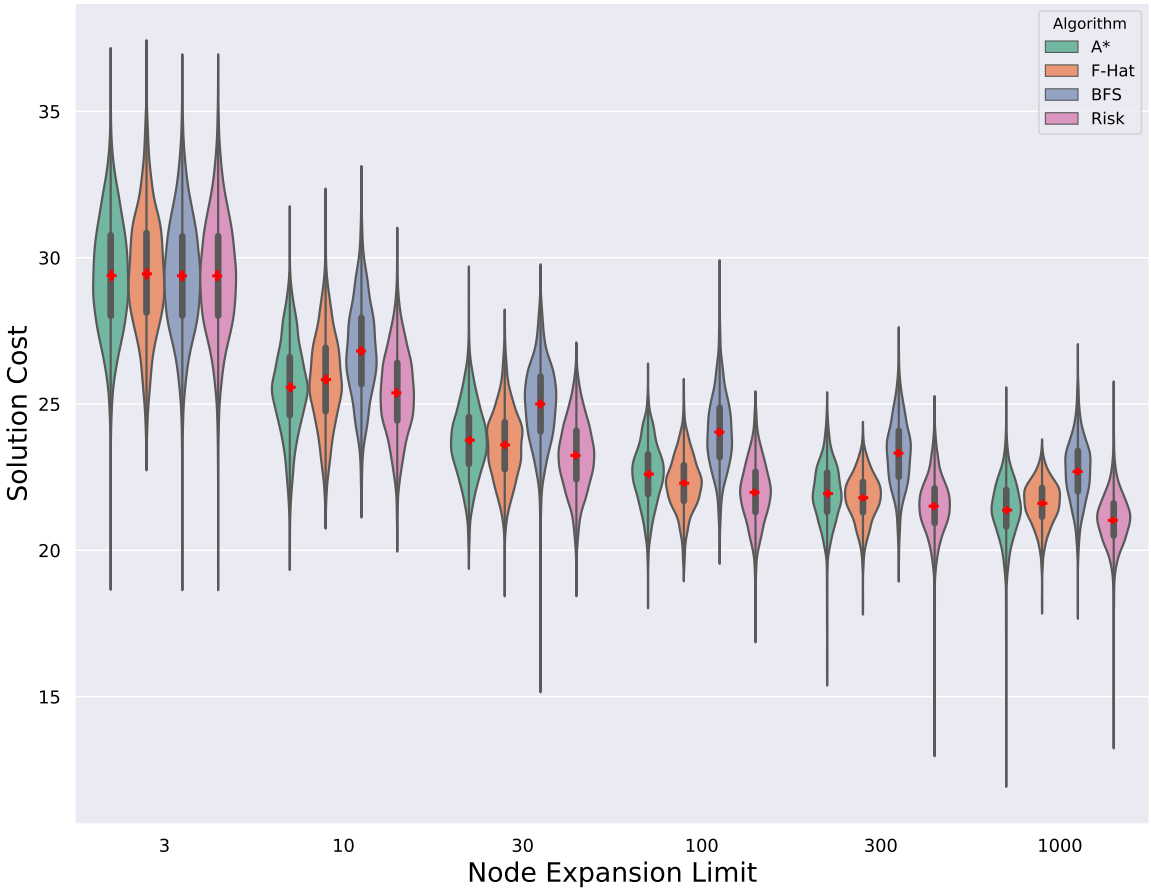

Figure 2-2: Expansion strategies on random trees. Violin plots show distribution of the data as well as a box plot with the median and quartiles. The red horizontal lines depict mean while red vertical lines show the 95% confidence intervals.

2.3 Implementation

The following section covers implementation details that are unique to topics discussed in this chapter. However, implementation details discussed in chapter 1 such as the random tree domain, beliefs using discrete distributions (specifically the squish operator which is utilized by risk based expansion), top level actions and their open lists, and heuristic error calculation are still applicable here. These implementation details remain unchanged for the expansion strategy experiments.

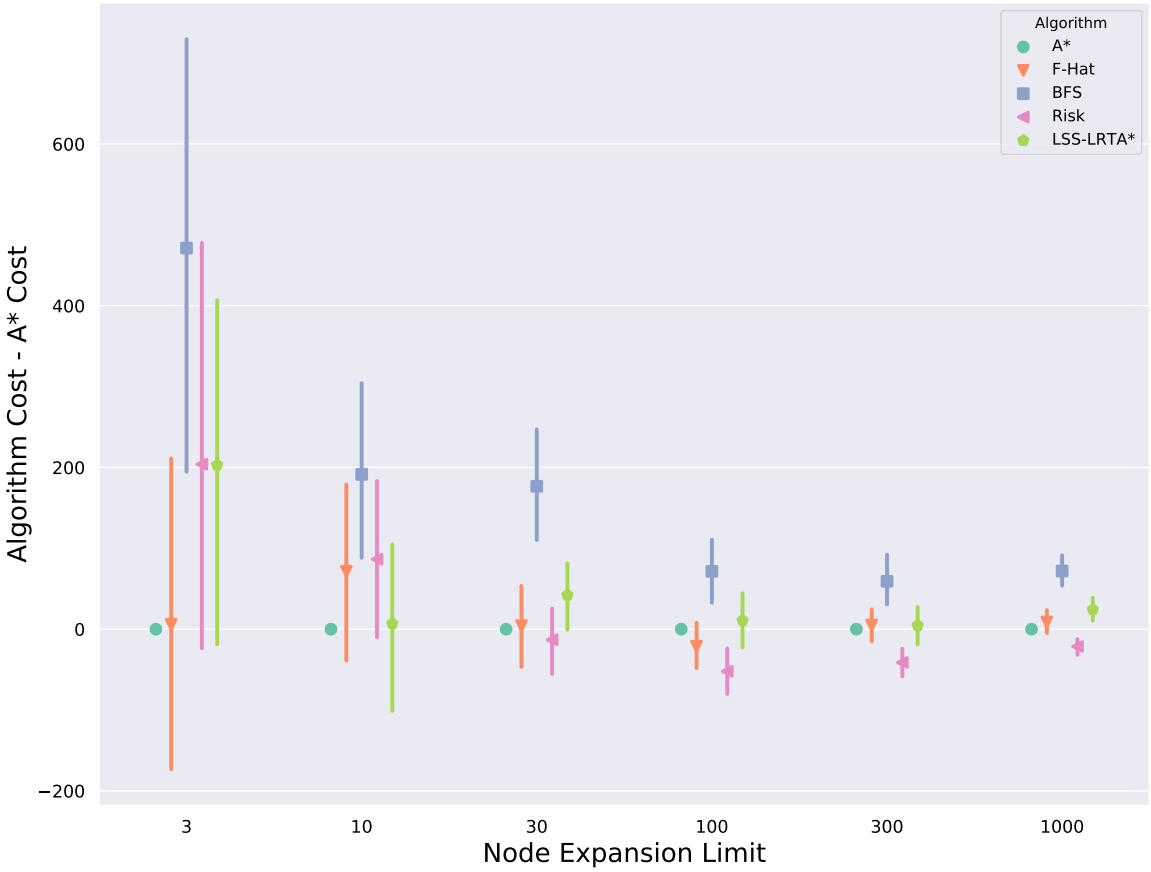

Figure 2-3: Expansion strategies on 15 puzzles.

2.3.1 Risk Based Expansion and Nancy

Nancy is our algorithm that utilizes risk based expansion and Nancy backups. Psuedocode is given below. The only functions that aren't expanded on below are learnHValues(), which is the typical reverse Dijkstra function used in LSS-LRTA* to learn heuristic values, and updateTLABeliefs(), which just pulls the belief of the best node on every TLA's heap up as the belief of that TLA, according to the Nancy backup rule. The function squishBelief() is also omitted, as it was explained in a previous section how beliefs are squished to simulate how additional search would impact the existing belief. It should be noted that the squishing of a TLA's belief is not a permanent operation, and is only applicable in a single iteration of the outermost loop in riskCalculation().

Nancy begins by using its first expansion to generate the top level actions and their beliefs, which are based on the beliefs of the immediate successors of the current state. From that point forward,

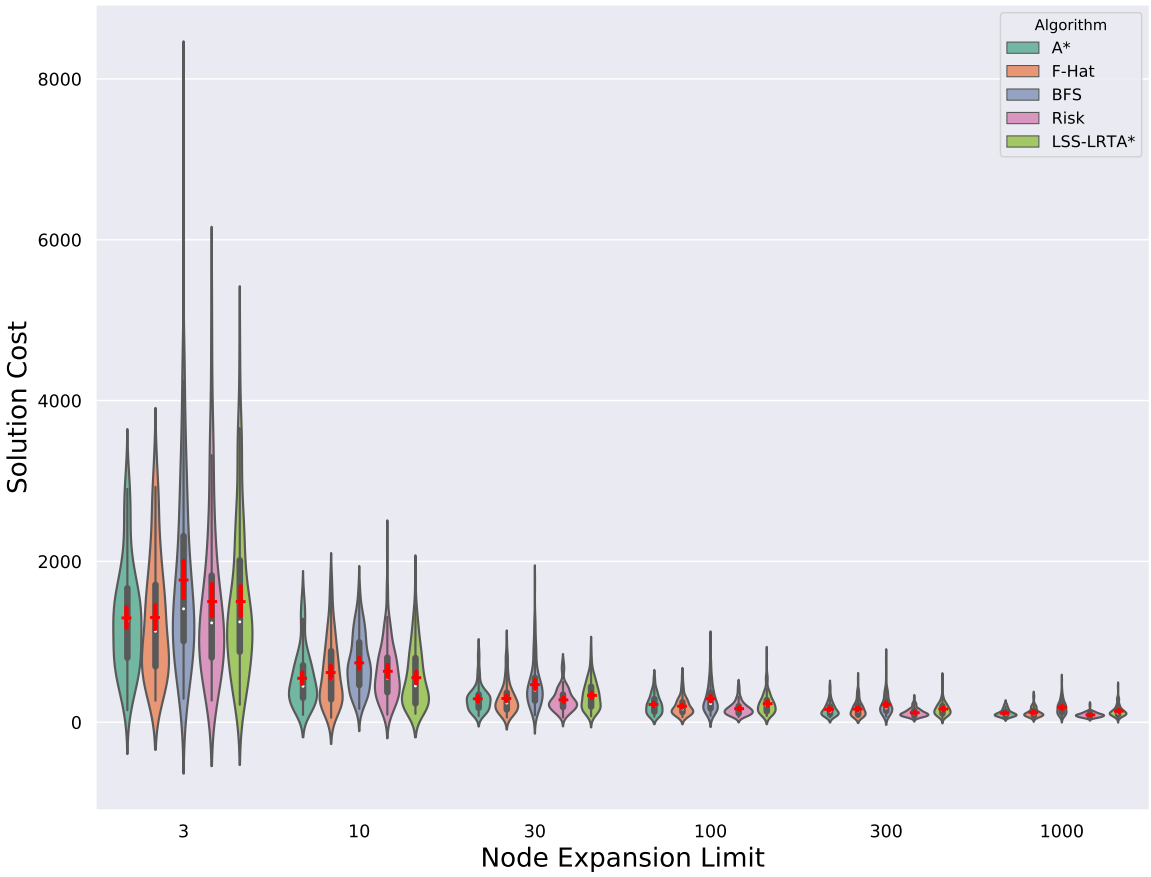

Figure 2-4: Expansion strategies on 15 puzzles. Violin plots show distribution of the data as well as a box plot with the median and quartiles. The red horizontal lines depict mean while red vertical lines show the 95% confidence intervals.

Nancy will expand nodes such that an approximation of risk is minimized, until the expansion or time limit runs out. When expanding nodes, Nancy will first update the beliefs of all TLAs, so that they reflect any changes made during the last expansion. It will then identify the TLA with the lowest expected cost, α, which is the TLA that would be executed according to the Nancy backup if search were to end at this moment.

Once α has been selected, Nancy will iterate over each TLA and squish its existing belief, as if the best node under that TLA had been expanded. The value of this expansion is calculated using risk, the expected loss if this node were expanded and α was not the optimal action to execute. Risk is calculated by performing numerical integration over every sample in α and every sample

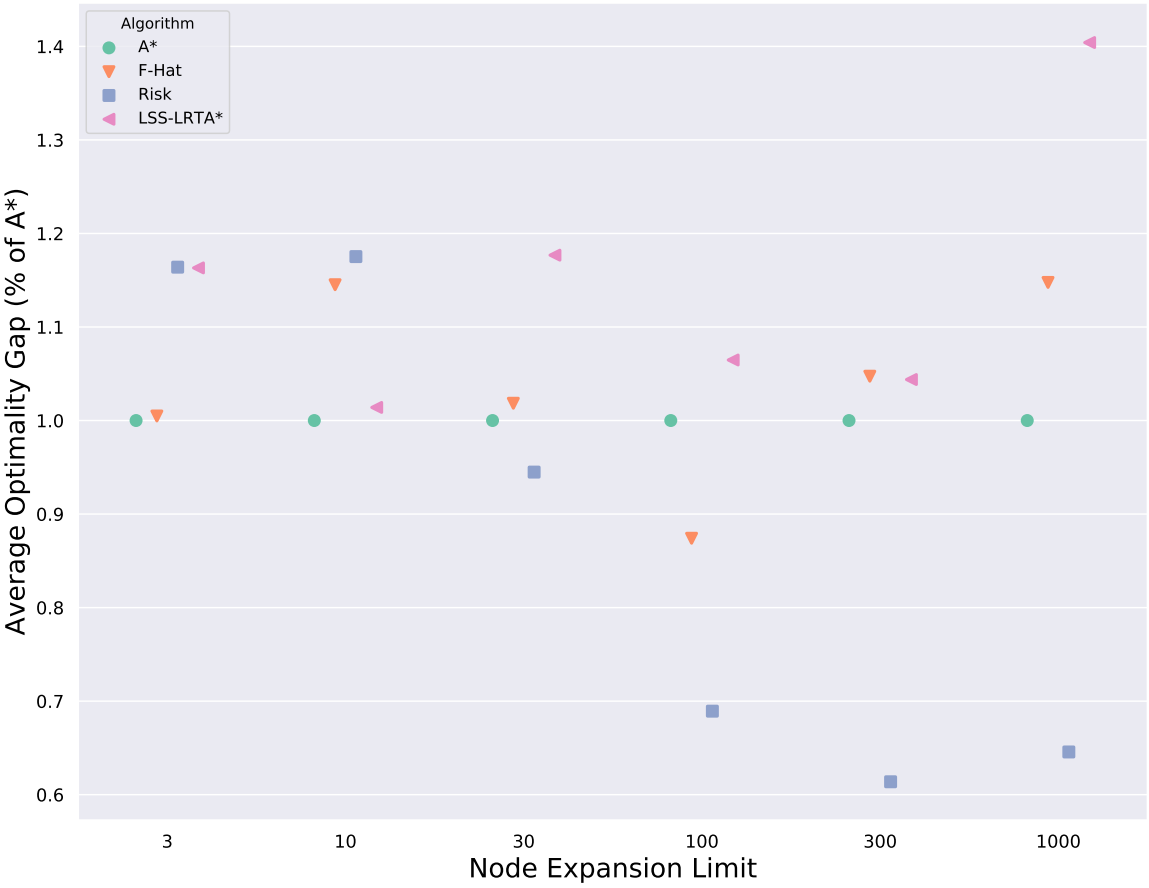

Figure 2-5: Optimality gap of expansion strategies on 15 puzzles as a percentage of A*'s.

with a cost less than the current sample in α in each of the β TLAs (TLAs other than α). This calculation is shown in lines 41 and 42 of Algorithm 2.

Once the risk of expanding each node has been calculated, the node with the lowest risk is actually expanded. This process is then repeated until the expansion or time limit is reached, or a goal state is expanded. Once the expansion phase ends, the agent performs Nancy backups and executes the TLA with the lowest expected cost. It transitions into the next state (using single-step commitment) and checks if the current state is a goal. If it is a goal, it returns a solution. Otherwise, the process will continue with the next expansion phase.

Algorithm 2 A search algorithm that uses risk based expansion and Nancy backups, which we have named Nancy for short

```
1: procedure NANCY(start, limit)
2:     cur = start
3:     while true do
4:         if isGoal(cur) then
5:             return solution
6:         closed = ∅
7:         tlas = generateTopLevelActions(cur)
8:         closed.insert(cur)
9:         riskExpansion(tlas, closed, limit)
10:        cur = nancyBackup(tlas)
11:        learnHValues()
12: procedure GENERATETOPLEVELACTIONS(cur)
13:    tlas = ∅
14:    for child in successor(cur) do
15:        heap = ∅
16:        tlas.insert(newTLA(child, belief(child), heap))
17:    return tlas
```

```
18: procedure RISKEXPANSION(tlas, closed, limit)
19:     expansions = 1
20:     while expansions < limit do
21:         updateTLABeliefs()
22:         chosenTLA = riskCalculation(tlas)
23:         cur = chosenTLA.heap.top()
24:         if isGoal(cur) then
25:             return
26:         chosenTLA.heap.pop(cur)
27:         closed.insert(cur)
28:         expansions++
29:         for child in successor(cur) do
30:             if !duplicate(child) then
31:                 chosenTLA.heap.push(child)
```

```
32: procedure RISKCALCULATION(tlas)
33:     alpha = lowestExpectedCost(tlas)
34:     for tla in tlas do
35:         squishBelief(tla.belief)
36:         tla.risk = 0
37:         for sample_α in alpha.belief do
38:             for beta in tlas s.t. beta != alpha do
39:                 for sample_β in beta.belief do
40:                     if sample_β.cost < sample_α.cost then
41:                         prob = sample_β.probability * sample_α.probability
42:                         tla.risk += prob * (sample_α.cost < sample_β.cost)
43:     return argmin_n(tlas[n].risk)
```

```
44: procedure NANCYBACKUP(tlas)
45:     updateTLABeliefs()
46:     return argmin_n(tlas[n].belief.expectedCost)
```

CHAPTER 3

Conclusion

Having reviewed the theory, experimental results, and implementation details surrounding this thesis, the only things left to discuss are the limitations and contributions of this work. Suggestions as to how these limitations may be overcome, as well as ideas for future work will also be addressed.

3.1 Limitations

While our experimental results are promising, it is important to recognize that the methods were compared using a fixed number of node expansions rather than CPU time. Although Nancy backups are not particularly expensive, our implementation of risk-based expansion numerically manipulates distributions and integrates over them. Such metareasoning overhead can be amortized by performing several expansions per decision, but it would be useful to engineer an optimized method. We believe that a lightweight approximation of risk-based expansion should be possible, perhaps along the lines of Thomspson sampling (Thompson, 1933) or UCB (Auer, Cesa-Bianchi, & Fischer, 2002).

Although we successfully adapted previous work on belief distributions for heuristic search, the issues of how best to represent beliefs and how to acquire them, perhaps through on-line learning, deserve sustained attention.

It was mentioned earlier that Nancy expands nodes such that an approximation of risk is minimized. It would be interesting for future work to explore how close to optimal Nancy comes in regard to risk minimization.

As Nancy was implemented with methods relying heavily on numerical integration, and was tested using node expansions as a limiting factor, it was not tested on more realistic domains. Ideally, the algorithm can be optimized so that it is reasonable to use CPU time as a limiting

factor, so then it can be tested in time critical domains such as traffic, where an agent has to find its way from one end of a grid to another while avoiding collisions with other moving agents.

3.2 Contributions

It is tempting to believe that, although reasoning under uncertainty is a popular topic in research on MDPs and POMDPs, it is not relevant for search in deterministic domains. However, this work illustrates how uncertainty can arise even in deterministic problems due to the inherent ignorance of search algorithms about those portions of the state space that they have not computed. Rationality demands that an agent select an action that minimizes expected cost, but this leaves open how to select nodes for expansion and how to aggregate frontier information for decision-making. We examined four backup rules, including two new ones, and a new expansion strategy that attempts to minimize risk. Experimental results on random trees and the sliding tile puzzle suggest that the new Nancy method provides results superior to traditional approaches. This work shows how an agent in a general real-time search setting can benefit from explicitly metareasoning about its uncertainty.

Bibliography

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, *47*(2-3), 235–256.

Bellemare, M. G., Dabney, W., & Munos, R. (2017). A distributional perspective on reinforcement learning. In *Proceedings of the Thirty-Fourth International Conference on Machine Learning (ICML-17)*.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

Cserna, B., Ruml, W., & Frank, J. (2017). Planning time to think: Metareasoning for on-line planning with durative actions. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS-17)*.

Frazier, P. I., Powell, W. B., & Dayanik, S. (2008). A knowledge-gradient policy for sequential information collection. *SIAM Journal on Control and Optimization*, *47*(5), 2410–2439.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions of Systems Science and Cybernetics*, *SSC-4*(2), 100–107.

Kiesel, S., Burns, E., & Ruml, W. (2015). Achieving goals quickly using real-time search: experimental results in video games. *Journal of Artificial Intelligence Research*, *54*, 123–158.

Kocsis, L., & Szepesvári, C. (2006). Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML-06)*, pp. 282–293.

Koenig, S., & Sun, X. (2008). Comparing real-time and incremental heuristic search for real-time situated agents. *Journal of Autonomous Agents and Multi-Agent Systems*, *18*(3), 313–341.

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, *27*(1), 97–109.

Korf, R. E. (1990). Real-time heuristic search. *Artificial Intelligence*, *42*, 189–211.

Mitchell, A. (2018). C++ source code for real-time heuristic search. `https://github.com/ajx256/thesis-real-time-search`.

Mitchell, A., Ruml, W., Spaniol, F., Hoffmann, J., & Petrik, M. (2019). Real-time planning as decision-making under uncertainty. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*.

Mutchler, D. (1986). Optimal allocation of very limited search resources. In *Proceedings of the Fifth AAAI Conference on Artificial Intelligence (AAAI-86)*.

O'Ceallaigh, D., & Ruml, W. (2015). Metareasoning in real-time heuristic search. In *Proceedings of the Eighth Symposium on Combinatorial Search (SoCS-15)*.

Pemberton, J. C. (1995). k-best: A new method for real-time decision making. In *Proceedings of the 1995 International Joint Conference on AI (IJCAI-95)*.

Pemberton, J. C., & Korf, R. E. (1994). Incremental search algorithms for real-time decision making. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS-94)*.

Russell, S., & Wefald, E. (1991). *Do the Right Thing: Studies in Limited Rationality*. MIT Press.

Spaniol, F. (2018). Planning under strict time limits. Master's thesis, Saarland University.

Thayer, J. T. (2012). *Heuristic Search Under Time and Quality Bounds*. Ph.D. thesis, University of New Hampshire.

Thayer, J. T., Dionne, A., & Ruml, W. (2011). Learning inadmissible heuristics during search. In *Proceedings of the Twenty-first International Conference on Automated Planning and Scheduling (ICAPS-11)*.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, *25*(3/4), 285–294.

Tolpin, D., & Shimony, S. E. (2012). Mcts based on simple regret. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI-12)*.